パソコンプログラマのためのクリエイティブマガジン

平成3年4月2日第三種郵便物認可　1992年7月1日発行(毎月1回1日)　第2巻第7号通巻19号

月刊プログラマーズ・ページ

Programmer's Page

MONTHLY MAGAZINE FOR PROGRAMMERS

1992 7 JULY

定価980円



特集

過激なプログラミング入門

ソフトとハード，両極からのアプローチ

付録:5"2HDフロッピーディスク

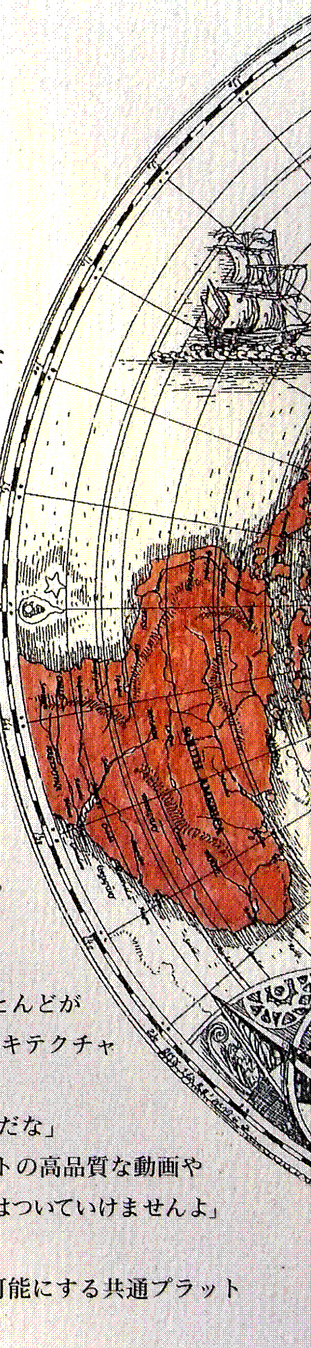
本誌掲載プログラム／綴じ込み付録テキストファイル（PC-9801・DOS/V エスケープシーケンスキャラクター覧）／クイズ・ランダムスクリーン解読ツール／3Dドライビングゲーム「Hover Run」／LOGOインタープリタ「Ashi LOGO v.1.0」(Cソース付)

SE
Shoeisha Co., Ltd.



世界標準物語　第三話

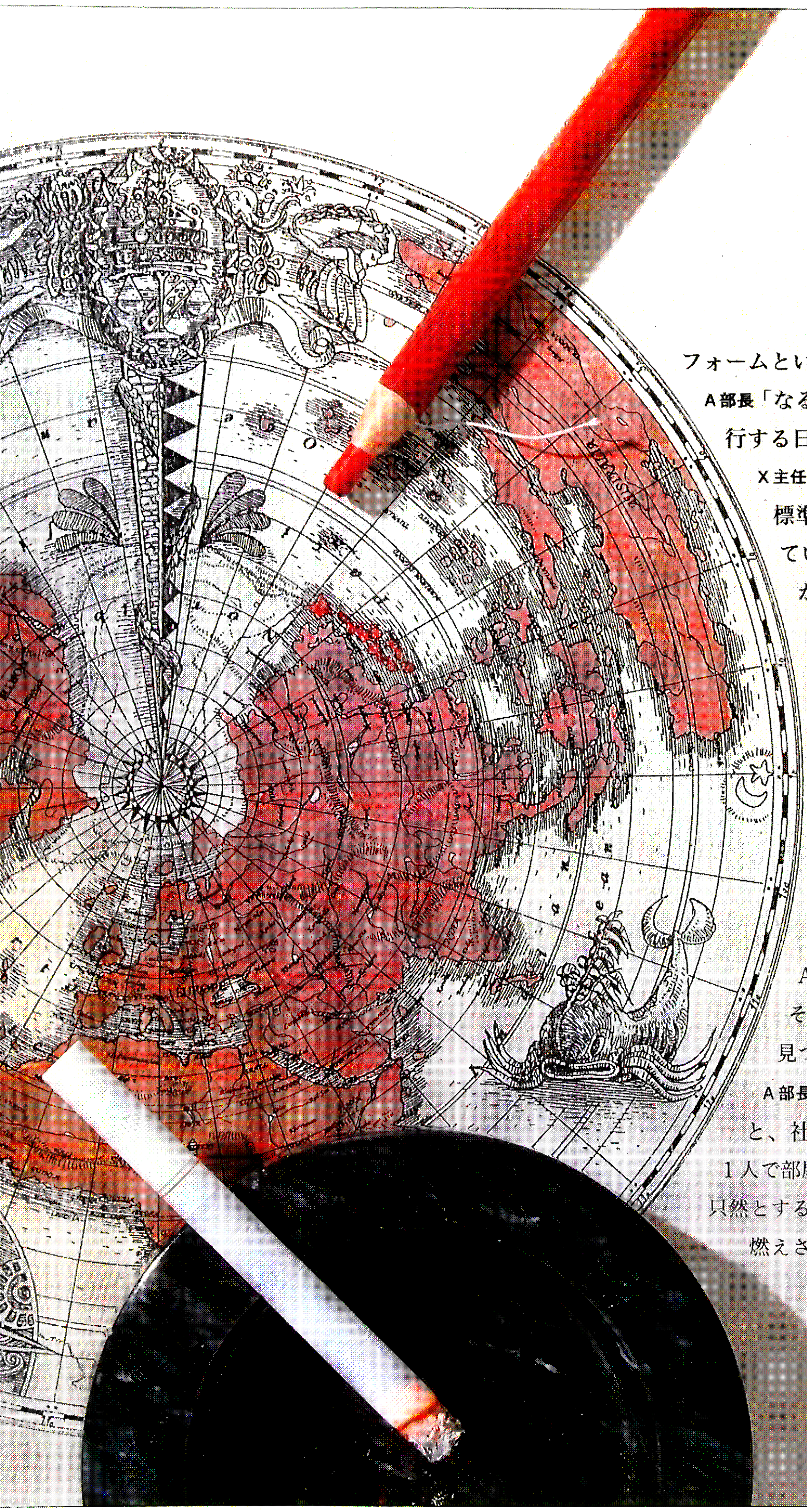
14本目の煙草



とある企業のオフィス。二人の男がテーブルをはさみ、真剣な顔で向き合っている。
テーブルの上の灰皿には、すでに12本の煙草の吸殻がある。
A部長「それじゃあキミはどうしても、AXがいいというんだね？」
X主任「はい、どうしてもAXがいいんです。今回だけは一歩も譲れません」
短い沈黙。A部長、13本目の煙草を取り出し、おもむろに火をつける。
A部長「しかし、うちの社内でAXを使っているのは、まだキミを含めて数人しかいないんだ。他のものが使いこなせるかね」
X主任「使いこなせるもなにも、操作自体は全く変わらないんです。違うのは、パソコン内部のアーキテクチャです。AXは、事実上の世界標準機であるIBMPC/ATと同じオープンアーキテクチャを採用しているので、世界に6万本以上といわれるIBMPC用ソフトがそのまま使えるんです。今、うちの社内で使っているパソコンは、日本では一応、標準機ということになってますが、閉ざされた、メーカー独自のアーキテクチャを採用しているので、そういうことはないんですよ」
A部長「IBMPCは、そんなに世界中で使われているのか？」
X主任、待ってましたとばかりに、一枚の世界地図をとりだす。
X主任「見てください。アメリカはもちろん、カナダ、メキシコ、ブラジル、イギリス、フランス、ドイツ、ソ連、中国…………。IBMPCとその互換機は日本以外のほとんどの国でトップシェアを占めているのが現実です」
X主任、つぎつぎに国名をあげながら、猛スピードでその国の位置を赤鉛筆で塗りつぶしていく。
A部長、思わず大きな溜息とともに煙を吐く。
A部長「うーん、これはすごい。日本はまさに世界の異端児ってわけだ」
X主任「MS-Windows、LAN、DTPソフトなど、現在我々が使っているソフトのほとんどが海外でIBMPC用として開発されたものです。日本語への移植も、AXなら基本アーキテクチャが同じなので即座に行えるというわけです」
A部長「GUI環境を実現するWindows3.0対応ソフトがすぐ使えるのは、確かに魅力だな」
X主任「それにAXはVGAに対応していますからね。これらの画像は、640X480ドットの高品質な動画やアウトラインフォントを扱えるVGAモードでなくちゃ、世界のマルチメディアの流れにはついていけませんよ」
A部長、煙草を揉み消し、身を乗り出す。
X主任「オープンアーキテクチャのAXこそが、ネットワークコンピューティングを可能にする共通プラット

世界の標準機を、日本の標準機に。
我々が、パソコンの新しい時代を拓きます。

Acer　Canon
MITSUBISHI　NCR　OKI　olivetti　SANYO



THE ARCHITECTURE EXTENDED-AX

フォームといえるんです」

A部長「なるほど、日本市場が海外市場と同時進行する日も近いわけだ」

X主任「部長、事実上の世界標準機が、日本標準機になる日が、もうすぐそこまで来ているんです。新しい時代に備えて、今が決断の時です」

再び沈黙。A部長、14本目の煙草に火をつける。

その時ドアをノックする音がして、部長の秘書が現われる。

秘書「部長、会議のお時間です。もう社長がいらしてますので急いでください」

A部長「しまった、もうそんな時間か。すぐに行くと伝えてくれ」

A部長、慌てて立ち上がりドアに向かう。そこで振返って、X主任の目をまっすぐ見つめて言う。

A部長「よしわかった。……AXしかないと、社長に言うことにしよう」

1人で部屋に残されたX主任は信じられぬ思いで呆然とする。が、すぐに満面に笑みを浮かべると、燃えさしの14本目の煙草を灰皿に押しつけた。

※MS-Windows 3.0は米国Microsoft社の登録商標です。
※IBM PC、IBM PC/AT、VGAは、米国IBM社の商標です。

CASIO　HEWLETT PACKARD　HITACHI　KYOCERA
SHARP　SONY　UNISYS

社名アルファベット順

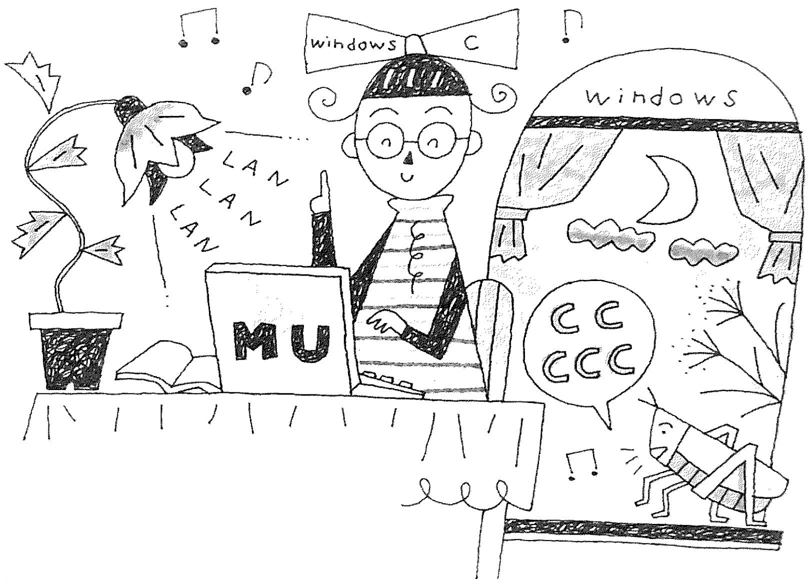
お問合わせ：株式会社 翔泳社
TEL. 03-3263-0447

次代のアドバンテージ、最新の技術をあなたに！

Microsoft® University

独学より大学。

マルチメディアのプラットフォームMS-Windows、高信頼性のMS OS/2。本格的なマルチタスクの実現、グラフィカルなユーザ・インタフェース(GUI)、LANマネージャによる高度なネットワーク処理など、技術者の創造力を刺激する環境を提供します。マイクロソフト・ユニバーシティは、その登竜門。ウインドウ環境、デバイスドライバの開発、ネットワーク処理など、アプリケーション・ソフト開発に欠かせない、ノウハウをレクチャします。入門から専門まで、充実したコースラインアップ。短時間で即開発に結びつく新しいコンピューティングスタイルを身につけて下さい。



Microsoft Windows Programming Environment
ウインドウズ・プログラミング・エンバイラメント

■**期間**：5日間 ■**受講料**：18万円(税別)

■MS Windows Ver3.0のアプリケーションを作成する方のためのコースです。Windowsプログラミングのポイントを重点的に解説、実習中心のコースです。

Microsoft Windows Applications Programming
ウインドウズ・アプリケーションズ・プログラミング

■**期間**：5日間 ■**受講料**：18万円(税別)

■MS Windows Ver3.0対応のアプリケーションを開発するプログラマや設計者のためのコースです。ユーザインタフェースやアプリケーションなど設計ポイントがマスタできます。

Microsoft LAN Manager Programming Environment
LANマネージャ・プログラミング・エンバイラメント

■**期間**：4日間 ■**受講料**：15万円(税別)

■マイクロソフトLANマネージャ(LM)用アプリケーションを開発する設計者のためのコースです。LMのアーキテクチャと特徴について解説します。

Microsoft LAN Manager Applications Programming
LANマネージャ・アプリケーションズ・プログラミング

■**期間**：5日間 ■**受講料**：18万円(税別)

■マイクロソフトLANマネージャ(LM)上で動作するアプリケーションの作成や移植するプログラマのためのコースです。

MS OS/2 Presentation Manager Programming Environment
プレゼンテーション・マネージャ・プログラミング・エンバイラメント

■**期間**：5日間 ■**受講料**：18万円(税別)

■MS OS/2プレゼンテーション・マネージャのアプリケーションを作成するための入門コースです。作成に必要となる基本事項を説明いたします。

MS OS/2 Presentation Manager Applications Programming
プレゼンテーション・マネージャ・アプリケーションズ・プログラミング

■**期間**：5日間 ■**受講料**：18万円(税別)

■MS OS/2 PMのアプリケーションを作成するためのステップアップコースです。経験者の方を対象に、より詳しく説明します。

MS OS/2 Programming Environment
プログラミング・エンバイラメント

■**期間**：4日間 ■**受講料**：15万円(税別)

■MS OS/2のアプリケーションを作成するために必要となる基本的技術を修得するコースです。本コースは、OS/2に取り組む上での基本コースとなります。

MS OS/2 Applications Programming
アプリケーションズ・プログラミング

■**期間**：5日間 ■**受講料**：18万円(税別)

■MS OS/2のAPIを使用し、マルチタスク、メモリ管理、プロセス間通信、ダイナミックリンクなどの機能を有効に活用する、実習を中心としたコースです。

MS OS/2 Device Drivers
デバイス・ドライバ

■**期間**：4日間 ■**受講料**：15万円(税別)

■MS OS/2のデバイス・ドライバの機能、構造を修得し、実際にデバイス・ドライバを作成するコースです。デバイスヘルパールーチンなども解説します。

C言語入門

■**期間**：2日間 ■**受講料**：6万円(税別)

■これからC言語を始めようとする方を対象に、C言語の基本的な文法からわかりやすく解説します。統合環境を持つQuick Cを使って理解を深めます。

C言語応用

■**期間**：3日間 ■**受講料**：9万円(税別)

■関数、ポインタ、構造体についてプログラムを作成しながら詳しく説明します。セミナは、最新のMicrosoft C Ver 6.0を使用して実施いたします。

●お問い合せは

富士ソフトウエア株式会社 MU教育センタ

TEL.03-5461-8296 FAX.03-5461-8469

日本電気株式会社殿の御協力によりPC-9800シリーズ対応コースも好評開催中!!

Microsoft®
マイクロソフト株式会社
〒160 東京都新宿区西新宿7-5-25 Kビルディング

FSI 富士ソフトウエア株式会社
MU教育センタ
〒108 東京都港区港南2-12-27イケダヤ品川ビル

〈資料請求番号001〉

まさにプロフェッショナル!!

Yoshimasa Electronic presents MS libraries Microsoft. Libraries

RS-232C Library　通信ソフトY-TERM装備　価格 23,000円 発売中

■**製品概要**　本製品は、C言語にてRS－232Cの各種制御を行うためのライブラリです。一般的な調歩同期式だけではなく、同期式についても完全にサポートされていますので、BSC等同期式手順を使うアプリケーションにも対応できます。また付属している通信ソフトY-TERMは、プログラム例としてだけでなく実際にパソコン通信等にもお使い頂くことができる機能を備えております。

■**主な特長**

●調歩同期式・同期式手順ともにサポート
●半／全二重制御、ハード／ソフトフロー制御のサポート
●拡張RS－232Cボード(PC－9861K)対応
●実用通信ソフトY-TERM装備
●Y-TERM・ライブラリともにソースコード完全公開
●組込みロイヤリティ無料

対応機種　NEC PC-9801シリーズ
対応言語　MS-C 5.1以上
Quick-C 2.0以上
TURBO C++1.0



■Y-TERMの主な機能
●モデムの監視機能 ●オートログイン機能 ●自動接続機能 ●テキストファイル送受信機能 ●XMODEMファイル転送機能 ●ユーザーファンクション

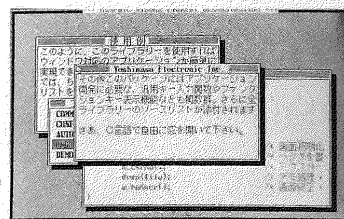
Graphic Window Library　グラフィック ウィンドウ ライブラリ　価格 23,000円 発売中

■**製品概要**　本製品は、C言語にてウィンドゥ処理を行うためのライブラリです。このライブラリを使用することにより、ウィンドゥ対応型アプリケーションを簡単に実現致します。ウィンドゥ間の切り替えや移動等、マウスに完全対応していますのでマン・マシン・インターフェースは、抜群の威力を発揮します。またグラフィックを使用していますので、一行のドット数が可変となり、ANKフォントも自由に設定可能です。

■**主な特長**

●高速グラフィックウィンドゥ　●ソースコード完全公開
●マウス完全対応　●組込みロイヤリティ無料
●汎用キー入力ルーチン装備

対応機種　NEC PC-9801シリーズ
(ハイレゾリューションモードは除く)
対応言語　MS-C 5.1以上
Quick-C 2.0以上
TURBO C++1.0



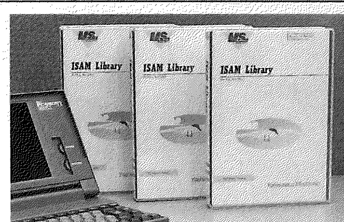
ISAM Library　アイサム ライブラリ　価格 23,000円 発売中

■**製品概要**　本製品は、C言語にてISAMファイル処理を実現するライブラリです。検索は、B－treeアルゴリズムを採用しているため、高速検索が可能です。当ライブラリにて作成のデータベースファイルは、dbaseⅢplusにて参照、変更可能です。

■**主な特長**

●B－treeによる高速アクセス ●ソースコード完全公開
●組込みロイヤリティ無料

対応機種　MS-DOSが動作可能な機種
対応言語　MS-C 5.1以上
Quick-C 2.0以上
TURBO C++1.0



JCA 手順 Library　通信ソフト JTRN装備　価格 46,000円 発売中

(RS-232C Libraryが含まれます)

NEWS

●PC-NET／2(オンライン・ユーザー・サポート・システム)遂に登場
快適な操作性を目指したOS／2対応のユーザー支援システムです。マルチ回線によるパソコン通信での製品案内や問い合わせ、さらにバージョンアップの情報交換など幅広い御利用が可能です。
'92年6月1日発売開始　全ソース付 198,000円

●弊社製品情報ネットワークシステム"Y-NET"開局中!!アクセス電話番号(03)5350-1785 USER_ID"GUEST"でどうぞ!

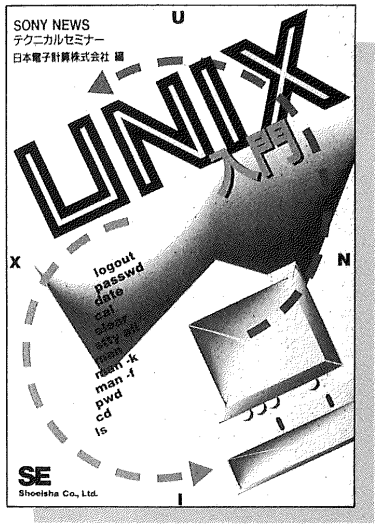
お求め方法　弊社製品は、パソコンソフト取扱店でお求めいただけますが、直接ご注文の際は右記までお問い合わせください。
なお、上記価格には消費税が含まれておりません。〈資料請求番号002〉

吉正電子株式会社 PC事業部
〒151 東京都渋谷区代々木1-58-10 第一西脇ビル
電話 (03)5371-3041(直通)

UNIX入門
―SONY NEWSテクニカルセミナー―

日本電子計算株式会社　編
A5判/237頁
定価1,800円

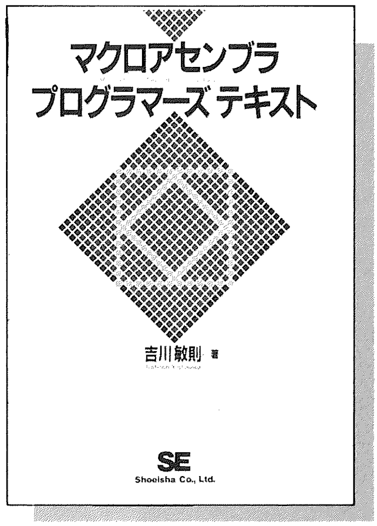
『ソニーのNEWSによる「UNIX入門」セミナーのテキストをベースとした入門書。UNIXのコマンド、エディタの使い方、日本語入力の方法、ネットワークの利用方法などを豊富な実例をまじえて解説。はじめてのワークステーションを利用するユーザ必携の書。』



マクロアセンブラ プログラマーズテキスト

吉川敏則　著
A5判/243頁
定価2,600円

『最低限必要なハードとソフトの知識から、データ演算の仕組みを踏まえた上で、アセンブラプログラミングの基本を丁寧に解説。アセンブリ言語に興味を持つ人や、これからアセンブリ言語でプログラムを作成する人に役立つ一冊。』



C文解釈
―「読む」ことからはじめるC言語―

豊国永健　著
B5判/183頁
定価2,000円

『C言語の文法書ではあまり取り上げられてこなかったわかりづらいポイントを、実際の応用プログラムを用いて解説。C言語の入門書とプログラムリスト中心の本との橋渡しになります。』



Programmer's Page

MONTHLY MAGAZINE FOR PROGRAMMERS

特集

過激なプログラミング入門

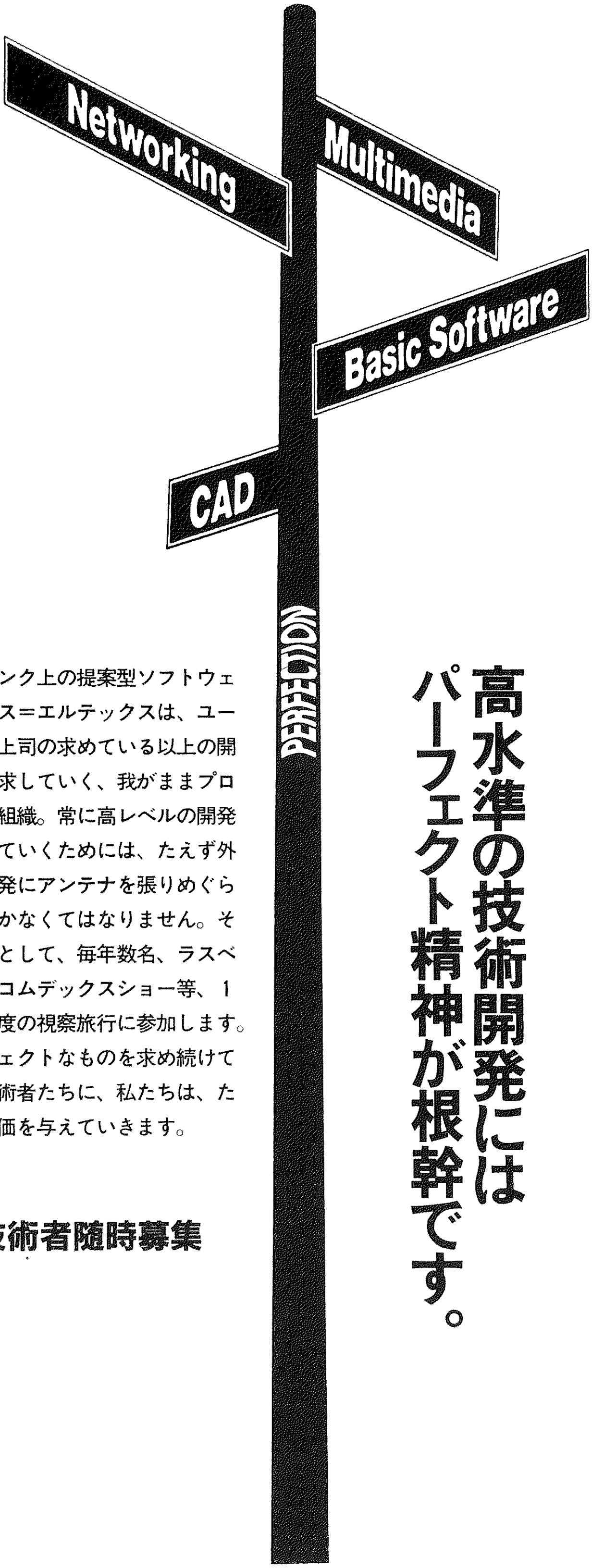
ソフトとハード，両極からのアプローチ

表紙イラスト：日暮修一
表紙AD：佐々木幹夫(SE)

広告索引　資料請求番号（掲載順）

人材募集広告

1992 JULY No.19



高水準の技術開発には
パーフェクト精神が根幹です。

ワンランク上の提案型ソフトウェアハウス＝エルテックスは、ユーザーや上司の求めている以上の開発を追求していく、我がままプロ集団の組織。常に高レベルの開発を行っていくためには、たえず外部の開発にアンテナを張りめぐらしておかなくてはなりません。その一環として、毎年数名、ラスベガスのコムデックスショー等、1週間程度の視察旅行に参加します。パーフェクトなものを求め続けていく技術者たちに、私たちは、たえず評価を与えていきます。

技術者随時募集

会社概要

ELTEX

設　立	昭和60年12月14日
資本金	1000万円
	1 ネットワークの端末システムの開発 2 CD－ROM、LDなど新しいメディアに対応したシステムの開発 3 OCD、CADなどのソフトウェアプロダクトの開発、販売 4 ビジネスパッケージの開発、販売
特　徴	大手クライアント向けにネットワークやマルチメディアなどのシステム開発を幅広く対応している。また、手書文字読み取りシステム「読造」*1、水道工事店向け「配管CAD」*1 などの自社ソフトウェア製品の開発販売や、電子回路CAD「Tango」シリーズ*2の日本語化など積極的な事業展開をしている。 *1　当社で開発し三菱系ディラーで販売しているプロダクト *2　米アクセルテクノロジ社の製品をPC9801への移植と日本語化したプロダクト、㈱ソーテック工人舎事業部で販売している。

募集要項

職　種	①システムエンジニア　②プログラマー
資　格	専門卒以上20～36歳の方　※C言語・アセンブラ言語経験者大歓迎　※未経験者でも好奇心・向学心の旺盛な方は歓迎します。
給　与	17万円（専門卒初任給）～46万円（36歳課長、諸手当含む） 24歳例：23万5000円（情報処理1種手当・住宅手当含む） 30歳例：31万6000円（中級SE、情報処理1種・住宅手当含む）
手　当	時間外手当、役職手当（5000円～6万円）、住宅手当（1万5000円）、資格手当（2種：5000～特種：2万円）、通勤手当
賞　与	賞与年2回（平成1年度実績年間5ヵ月）、決算賞与年1回（平成1年度実績1ヵ月）
勤務地	本社（横浜駅近く） ◇派遣・出向はありません。
勤務時間	フレックスタイム制（コアタイム10:00～16:00）試用期間中は標準時間9:00～17:45
休日休暇	完全週休2日制（土・日）、祝日、年3回連休（GW・夏期・年末年始）、有給・特別・休暇
福利厚生	各種社会保険完備、退職金制度、財形貯蓄、社員旅行、海外研修、定期健康診断、社員寮、スポーツクラブ法人会員
応募方法	電話連絡の上、履歴書（写真貼付）、職務経歴書を持参し来社ください。郵送も受け付けます。
連絡先	〒220 横浜市西区南幸二丁目20番5号　東伸24ビル5階　℡(045)322-9667　総務課

採用HOT・LINE

TEL.045-322-9667

株式会社エルテックス

ソフトとハード，両極からのアプローチ

過激なプログラミング入門

●特集

一般にプログラミング入門というと，C言語ならば“Hello World.”，FORTRANならば三角形の面積を求めるプログラムからはじめるものと相場が決まっている．しかし，ほんとうにこのようなやり方でプログラミングが身につくものだろうか？　はっきりいって疑問である．少なくとも，このようなプログラミング言語の文法から入るやり方だけでは，プログラミングの本質も，コンピュータの本質も身につけることはできないだろう．そこで，現実の問題をプログラムへと結び付ける，プログラミングの最も重要な課程であるプログラム設計と，コンピュータの本体であるCPUのマシン語命令がどのような意図に基づいて設計されているかを学ぶことから，ちょっと“過激な”プログラミング入門を試みた．

特集
過激なプログラミング入門

プログラミングをやってみようと思ったとき，たいていの人が手にするのがC言語あるいはBASICなどのプログラミング“言語”入門書であろう．しかし，これがそもそもの間違いなのである．日本語の文法の勉強をしても小説が書けるようにはならないように，C言語の文法を学んだところでプログラムが書けるようにはならない．プログラミングを身に付けるためには，プログラミング言語を学ぶ前に，プログラムの書き方を学ばなければいけないのである．プログラミングの本質をゼロから学んでみようではないか．

～プログラムの川を渡るために～

ゼロからはじめるプログラミング

株式会社ジャステックラボ
加野島英渡＋山本博昭

このページを開いている読者のほとんどがパソコンの使用経験をもっていることと思う．その中で実際に，コンピュータのプログラムをなにか書いたという経験をおもちの方はどのくらいいいるのだろうか．

この記事のタイトルである「ゼロからはじめるプログラミング」というフレーズに多少なりとも興味をもっていただいた方の中には，まだ1度もプログラムを書いたことがない方もいるのではないだろうか．また，何を身につければプログラミングというものができるようになるのか知りたいと思われている方もいるかもしれない．

今回の特集では，実践的なプログラミングテクニックや具体的なコンピュータ言語によるプログラム書法は出てこない．プログラムを書くために，エディタやコンパイラを使う前の「どうしたらプログラムを書く気になるのか」という問題を中心に進めていくことになる．パソコンを使うからには，1度は自作のプログラムを走らせる快感を知っていただきたい．本章は，プログラミングの道をめざしてこれから歩き出そうという人々に，第一歩の足をいかにして上げるかを提言していきたいと思う．

1. 人間界の巻

1-1 道具で遊ぶ動物・人間

かの名画「2001年宇宙の旅」では，人類が道具らしきものを使いだす印象的なシーンがあった．ほかの生物と比較して際立った肉体的な長所をもたない人類が，今日まで大きな顔をしてこられたのも，道具を作ってそれを使うことに長けていたからであることは，誰の目にも明らかである．道具なしには生きていられなかった人類は，いつしかこのパートナーに愛着を覚え，生きるための手段以上の感情を抱きはじめた．それが昴じていって，やがて「道具マニア」が誕生してきた．高速かつ大量移動の画期的手段として登場した鉄道や自動車，命を守り食料を調達するためのナイフや剣，こういった道具を種の存続と関係なく愛好する人々がいることはご存じであろう．

そして，ご存じパソコンを趣味にもつ同志がいる．中には「個人的にはコンピュータなど大嫌いだが，仕事だからしょうがなく使っている」といって顔を曇らせる不幸な人もいるのだろうが，この本を読んでいる読者に限ってそんなことはないと安心して断言しよう．コンピュータほど奥が深くて長い間楽しめる趣味はそう多くない．楽しみぬいた果てにこれで飯を食う（禄を得ているという意味である．決して箸や茶碗のかわりにしているわけではない）ことになってしまった筆者がいうのだからまちがいはない．

一口にコンピュータの趣味といっても，音楽や画像，ゲームに通信と守備範囲がおそろしく広い．しかし何といってもコンピュータで遊ぶ最大の醍醐味は「プログラミング」である．不幸にしてプログラミングの経験がない人，とてもそんな難しそうなことは無理だと決めてかかっている人，何度かチャレンジしたが歯が立たなかった人，さまざまな「プログラムしていない歴何年」の層があろうが，この特集ではそういった人々に今1度メッセージを伝

えたい．進入経路を少し変えてみれば，プログラミングへの道は意外に近くにあるものなのだ．

1-2 使われる人・使う人・使いこなす人

ビデオレコーダーなどのAV機器を所有している読者は多いと思うが，いま使っている機器を購入したときに，その組み立てや接続は自分で行っただろうか．どうもあの手のものはよくわからんといって，配達してもらった係員にすべて任せてしまうという人もいるようだが，実は道具を遊ぶための第一歩は箱から出す瞬間から始まることは，知る人ぞ知る事実である．

こうした機器の接続方法は，ただ1つの事実を把握してしまえば簡単である．それは「出るところと入るところを結ぶ」ということである．怒ってはいけない．たとえば，図1のようにカセットデッキとアンプを接続する場合には，カセットデッキの出力（カセットデッキから出る信号であるから，カセットテープの再生信号を意味する）は，アンプのカセット入力端子に入る．逆にカセットデッキの入力（これはカセットテープへの録音信号になる）は，アンプの録音用出力端子と繋がる．双方の端子がわかれば，あとはその端子に物理的に装着できるコードを探せばよい．

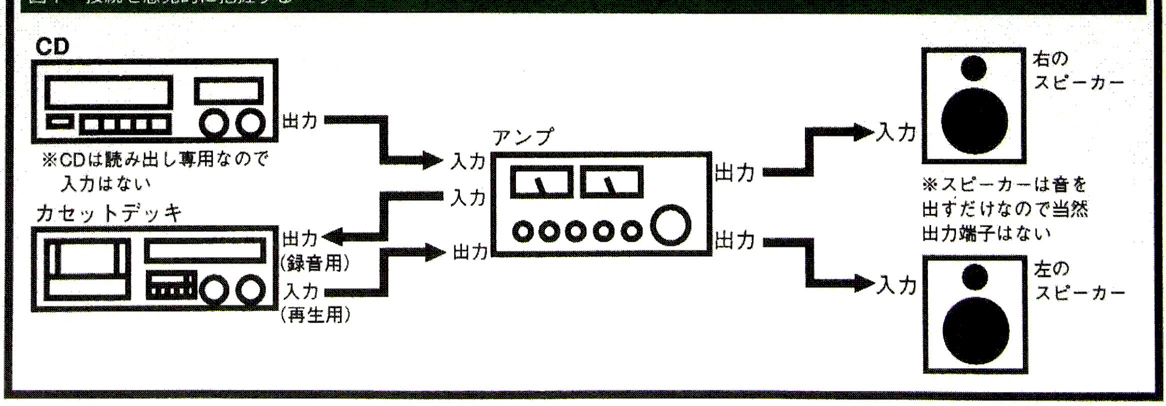
CDプレイヤーやスピーカーなどの機器も同様に接続していくわけだが，ここで1つ考えていただきたいのは，たとえばスピーカーというのは「音を出すための機器」だということである．カセットデッキが録音・再生という入出力をもつのに対し，スピーカーは音を出すための信号を受け取る入力端子しかついていない．これをアンプ側からみれば，スピーカー出力端子はあっても，スピーカー入力端子はないということである．同様に，書き込むことのできないCDも，アンプ側にはCD入力端子しかついていないはずである．

コンピュータと関係のない話が長くなってしまったが，要するに機器の性質をもとに信号の流れを把握すれば，接続は難しくないということである．信号を「データ」という言葉に置き換えれば，コンピュータの接続も同様である．ディスプレィは画像を表示するための信号データを受け取るだけなので，本体から見ると出力端子しかない．逆にキーボードは，人間が入力したデータを本体に伝えるだけなので，本体側には入力端子しかない（正確を期すと，この2つの機器にも機器を制御するために入出力端子を両方備えている機種もある）．

もっともほとんどのコンピュータは，周辺機器を接続するための端子に工夫が凝らされていて，端子の形を変えたり，上下逆に入れようとしても入らないようになっていたりして，キーボードを繋ぐ端子にディスプレィを繋いでしまうなどということは起こらないようになっている．

プログラミングというのは，自分が使っているコンピュータをコントロールしてしまう手段である．そのためには，そのコンピュータがどういう機構で動いているのかを知っていたほうが「いい」．コンピュータ理論やハードウェアの構造を知りつくせというのではない（もちろん，そうしたほうがより好ましいのだろうが）．その第一歩が，キーボードから入ってくるデータが内部の処理を経て，ディスプレィに表示されたり，ディスクに書き込まれたりする経路を感覚的に摑んでおくことである．自分がこれからコントロールする機械のことを感覚的に把握していく姿勢があれば，引っ越しのたびに電気店から人を呼んで配線をお願いしなく

図1　接続を感覚的に把握する



プログラム開発用語集
アニメート　デバッガでプログラムを実行しながら，同時にソースファイル中の対応する行を表示する機能

てもいいというメリットも付随してくるであろう．

1-3 コンピュータの使い方をデザインする

プログラミングへの道の第一歩は，コンピュータの使い方の意識を変えていくことである．先ほど書いたように接続の理屈を考えてみても，コンピュータといえども魔法の箱ではなく，1つ1つの動作を観察していくととても単純なものなのである．その観察をすることなしに「まるで魔法だ」と思ってしまえば，プログラミングへの入り口を開くことはできない．各周辺機器のデータの流れを感覚的に把握することによって，接続方法も理解できてくるように，ソフトウェアの動作もただ漠然と使っているだけではなくて，内部でどのような処理が動いているかを考えていく姿勢が身につけば，おのずと使い方も変わってくるだろう．

現在のパソコンでは，プログラムはシステムによってメモリ上に読み出され，そこで実行されていく．たとえばワープロソフトは図2のように，キー入力を受けつける部分や画面表示をする部分，ディスクの読み書きや文書データを管理する部分などからなっていて，使う者の操作によってそれぞれの部分が対応して動作していく．起動して，ディスクから文書ファイルを読み込み，手直しをして保存するという操作をした場合に，どの部分の処理がどのように動いているか，またデータはどの様に処理されていくのかを考えてみよう．

文書ファイルを読み込むということは，ディスク上のファイルから本体のメモリ内のどこかに（一般に編集バッファと呼ばれる）データを複写することである．同様に，文書を保存するためにはメモリ上で編集中のデータをディスクに書き出すことである．普通のメモリは，電気的な原理でデータを保存しており，電源を切ると内容が失われる．それに対してフロッピーディスクやハードディスクなどは磁気的な原理でデータを保持するので電源を切ってもデータが消えることはない．そのかわり，メモリは磁気記憶に比べて処理速度が桁違いに速い．ここまでの事実を総合して考えれば，快適な操作を行うために編集中のデータはメモリ上に置かれること，編集中に電源をいきなり切ってはいけないことなどが自ずから明らかになってくると思う．

このように，プログラムの処理やデータの流れを感覚的に把握していくとコンピュータを使うときの姿勢も変化してくる．これがプログラミングを実践していくときも大切な要素になっていくのである．

「プログラミング」という言葉は，最近ではコンピュータ以外の分野でもよく耳にするようになってきた．代表的なのは，ビデオの録画予約やオーディオ機器での演奏や録音動作の設定である．ただ，コンピュータの「プログラミング」が通常のビデオの録画予約と違うのは,「これがこうなったらこうする」という条件の判断とその結果による次への選択枝を用意できることである．図3に例を書いてみたが，単純なタイマセットには条件判断がない．それに対して，プログラミングは判断材料によって複数の選択枝を用意しておき，状況に応じて適応する処理が変わってくる．

そして，どういった場合にどのような選択枝を用意するかをデザインすることが，実はプログラミングの中核なのである．そういった意味でいうと，プログラムとはよく入門書の最初に出てくるような「printf(" Hello");」のことでは断じてない．ある目的のために，いろいろな場合を想定し，それに対応する処理を考えていく．この時点での想定や対応の詰めが甘いとよいプログラムはできない．とくに異常時の処理，俗にいうエラー処理をきちんと考えているかがプログラムの善し悪しを左右することになる．プログラムを書いていくと気がつくが，全体に占めるエラー処理の割合は非常に大きい．

図2　パソコンの中でのワープロソフト動作の例

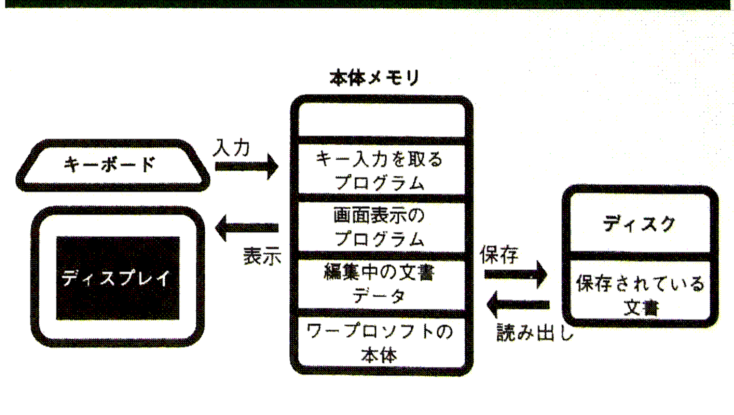
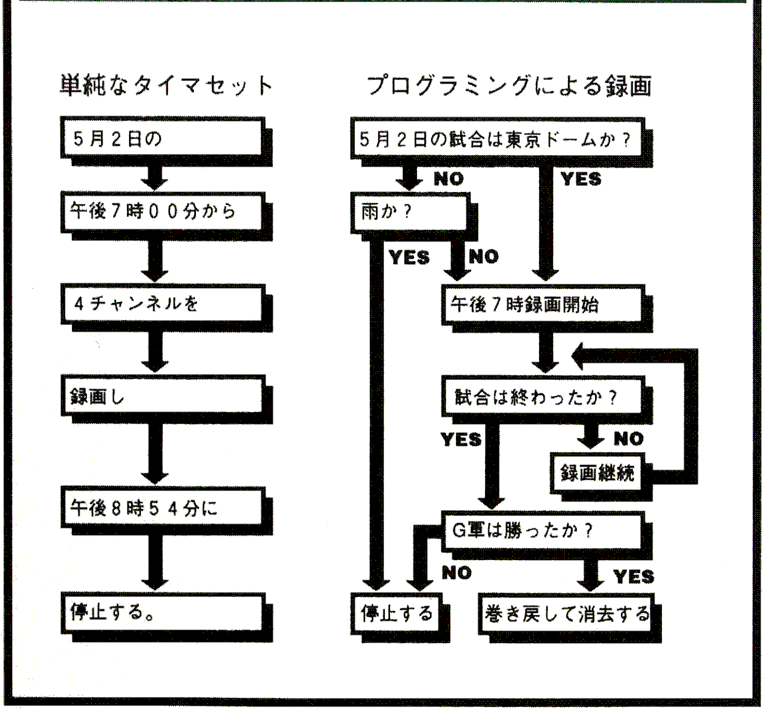


図3　プログラミング予約



このように，プログラムは考えられうるさまざまな状況を人間が想定して，その対応を計画していく手段である．その想定と対応によってコンピュータは，外から見るとまるで知能をもった機械のようにふるまうのだ．何でもできるようにみえるコンピュータが，実は人間が何かを用意してやらないと何もできない機械であるというのは紛れもない事実である．

1-4 プログラミングをしない・できないケースの観察

ここで,「プログラムをしたことがない」「やってみたことはあるが，書くことができなかった」という2人の人物を紹介しよう．知り合いを引き合いにだすのは少々気が引けるのだが，原稿のためなら何でもするという筆者のような友人をもってしまったことをあきらめていただくとしよう．

両名とももう数年来のパソコンユーザーで，ほぼ毎日何がしかの用途にパソコンを使っているにもかかわらず，プログラミングをしたことがない，あるいはしようと思ったができないでいるというケースである．

◎Y田K男氏（37才，中堅繊維メーカー総務課勤務）

3年前に購入した16ビット機を使っている．使用目的は，蔵書の目録データベースや仕事関係の住所録，将棋のゲームソフトなど．ワープロソフトは年賀状を作るときくらいしか使わない．当然職場にもパソコンがあるが，なぜかY田氏は個人でパソコンを所有していることを会社では秘密にしているそうである．データベースソフトの簡易言語は少々使えるが，コンパイラ言語などはまったく経験なし．やってみようと思ったこともないそうである．

・プログラムを書こうと1度も思ったことのない人の分析

Y田氏は，自分でも文系の出身であるために機械に強いとは思っていない．電化製品の調子が悪いときなども家族からはあまり頼りにされていない．3年前に「パソコンを買う」と宣言したときの家族の驚いた顔は今も鮮明に覚えているという．

そもそもY田氏がパソコンを買おうと思いついたのは，週刊誌で読んだ「個人データベースで情報化時代を先取りしよう」という特集記事で，住所録や蔵書録を作っておくとたいへん便利だという話を読んだためであった．むずかしい数式が出てこないことに安心したY田氏は，書店で買い求めた参考書をもとに，コツコツとデータを入力していった．3年たって数百冊になった蔵書データベースは，実用もさることながら，Y田宅を訪れる知人たちに披露されて大いに効果をあげている．Y田氏は，このシステムについて今のところさしたる不満もないと語る．速度が少し遅いと感じることもあるが，これは我慢すればよいのだし，ファイルの操作がめんどうでもとにかくコツコツとやっていれば，いつかは終わるものだから……ということであった．

◎M山S也君（21才，学生）

つい2か月ほど前に最新鋭の486機に買いかえた．ただし，今年いっぱいローンがあるのでアルバイトは欠かせない．MIDIやグラフィック，パソコン通信といろいろやってみたいことはあるのだが，周辺機器やモデムなどにまで予算が回らず，実践に移せないでい

プログラム開発用語集
インクルード　#include前処理指令などでヘッダやソースファイルをプログラムに挿入すること．

る．現在のところは，レポートを書くためのワープロソフトがおもな用途である．

実はM山君は2度ほどプログラミングにチャレンジして挫折している．1度目は，最初にパソコンを買った高校時代である．本体に付属していたBASICを使って何か作ろうと思い，画面に円を描くところまでいったが，これが実際のプログラムとどう結びつくのか想像がつかないまま一週間ほどで中断してしまった．そして2度目は去年の夏．酒に酔ったいきおいで友人に買うと宣言したCコンパイラを使って画面に「Hello」と表示したところで挫折という戦歴である．

・1度は書こうと思った，書いてみた人の挫折の分析

M山君がプログラミングをしようと思いたったのは，「何となくかっこよさそう」という理由からであった．プログラミング言語のマニュアルを最初に開いた瞬間から，頭の中は自作のプログラムを友人に見せている自分の姿でいっぱいだったそうだ．とにかく何か自慢できるものをと，意気込んではじめたのはいいが，画面にぽつんと丸が出たり，「Hello」などという字が出たりしても，コンピュータプログラムの何たるかを知らない人はそれほど感動してはくれないだろう．

自慢している自分の姿を夢見たまではいいが，M山君の敗因は，その想像上のコンピュータで動いている肝心のプログラムが何ひとつイメージされていなかったということである．どんな腕の立つプログラマでも，「何かすごいものを書こう」などという発想だけでプログラミングができるはずがない．どんなものがすごいのか，具体的にはどんなプログラムを作ることにするのか，そういったプランニングを何もしないまま断片的な知識を収集したとしても，そこから先には進めない道理である．

1-5 皆目見当がつかない，まったく歯が立たない訳

・どうしてプログラムを書いてみようと思わないか．

プログラミングへの道を進むきっかけは，疑問をもつことである．どうしてこのワープロソフトはこんなに遅いのか，どうしてファイルをコピーするのにファイル名を一字一句正確に入力しなければならないのか，どうしてたった1文字タイプミスをしただけなのに最初から入れ直さなければならないのか．疑問をもたずに「こういうものなのだ」と我慢してしまえばそれまでである．疑問をもつこと，そしてどうしたらよいのかを考えることによって，プログラミングの道は始まる．

Y田氏の場合は，疑問をもつということに今日まで無縁であった．最初は参考書に書いてあるとおり，一字一句正確に操作をしてみて，こつこつとデータを入力する毎日であった．多少操作がめんどうだと思っても，処理が遅いと思ってもすべて「コンピュータというのはこういうものなのだ」というコンピュータ側にとってみればありがたいのか迷惑なのかわからない理解の仕方をつらぬいてきた．Y田氏はデータベースソフトの「簡易言語」の経験はある．これとてプログラムの一種には違いないのだが，残念ながらこの経験もY田氏の無類の勤勉さとものわかりのよさによって，参考書のサンプルを正確に入力したのみにとどまっている．もう少しY田氏に欲があれば，「ここは自分はこのほうが使いやすい」という発想も出てきたはずなのだが…….

・どうしてまるっきり手も足も出なかったのか．

どういう動機であれ，1度はプログラムというものに挑戦して，ものの見ごとに挫折したという読者もいるのではなかろうか．せっかく買ったコンパイラソフトが引き出しの中で眠りについているとか，入門書を買い込んではじめてはみたものの「Hello」を画面に出したきりといった体験談は筆者の周辺でもときどき聞かされている．

プログラミングに挫折する大きな理由の1つに，普段使っているワープロや表計算などのソフトウェアと，これから自分がやろうとしているプログラミングとの間に，大きなギャップを感じてしまうという事実がある．画面に100回「こんにちは」と表示したところで，こんなことを続けていてもとても実用的なプログラムにはなりそうにもないし，データベースを作ろうなどと考えてはみても，いったい何をどうすればよいのか見当もつかない．つまるところ画面にいろいろメッセージを出したり，丸や三角を描いてみたところで中断ということになる．M山君がまさにその典型である．

やみくもにプログラムの文法や関数などを覚えても，プログラムを書けるようにはならないだろう．なぜならばプログラムというのは，数式や表示命令を羅列することではなく，論理を記述するものだからである．人間が論理的な思考をしようとするとき，その思考の対象となるべき具体的な題材がな

プログラム開発用語集
インデント　行の先頭にタブや空白を入れること．プログラミングスタイルのひとつでソースファイルでプログラムの流れを判断しやすくする．

いと非常に困難である．それゆえに「なんとなくプログラムというものを覚えたい」というスタンスではじめたプログラミング入門は，実を結ばない可能性が強い．

・プログラミングのはじまりには何を覚えるべきなのか

実際の話，プログラミングをはじめたばかりの人がワープロやデータベースのソフトウェアを作成するというのは無理なことである．どうして無理なことなのだろうか．コンピュータの知識があまりないからか，コンパイラ言語の予約語を覚えきっていないからか．いちばんの理由はそこにはない．言語のテクニックとかハードウェアの知識なども最終的には必要になってくるが，日本語を読み書きできる人がすべて小説を書けるわけではないように，こうした情報をもっていればプログラムが書けるということにはならない．

最初にしなければならないトレーニングは，自分が何をコンピュータにさせたいのかを明確にして，それを現実化していくための過程を構築することである．この作業の進め方を身につけないで，C言語だ，アセンブラだ，OOPだと頑張ろうとしてみても土台のないところに家は建たない．これからプログラミングをしてみようというからには，まず「自分がコンピュータに何をさせたいのか」を最低1つはもつことが条件である．

そうはいっても，今のところとくに要望はないという場合もあるだろう．この後の章ではそうした場合に「何をさせたいのかをいかにして思いつくか」というテクニックもいくつか紹介することにするので安心して読み進んでいただきたい．

・コンピュータにプログラムを実行させるということはどういうことなのか

結局のところ，プログラムというのはコンピュータに対する依頼事項を整理したものである．この「整理」が肝心なところで，人間ならば思いついたままいいつけたり，走り書きのメモのようなものを渡したとしても，相手側で判断して用が足りるということもあるが，コンピュータが相手ではそうもいかない．こちら側ですべて整理して，筋道のたった矛盾のない依頼をしないと動いてくれないのである．

とてつもなくわがままな話であるが，これだけのことを人間に要求するだけあって，いったん動いてしまえばコンピュータはあきれかえるくらい従順に確実に依頼をこなしていく．さっきまでワープロだったりゲームだったりしたコンピュータがその時点で自分が依頼したことを懸命に実行してくれるのだ．この事実を一回経験することがプログラミングの道にはまり込む起爆材となるだろう．道のりはまだかなり遠いが，頑張っていこうではないか．

1-6 コミュニケーションを習得するコツ

ビジネスマンたるもの英語の1つも話せないと，これからの世の中は乗り切れないといって英会話スクールへ通う人も多いと聞く．果たしてどのくらいの割り合いの人が目的を達成して「国際人」の仲間入りをするのかは定かではないが，もろくも夢破れて……という結果もないとはいえないだろう．

以前，英会話スクールで習うくらいなら単身アメリカに渡って半年も暮らしていたほうが早く話せるようになる（ただし，無事に帰って来られたら）という乱暴な話を聞いたことがあるが，なるほど日本に住んでいては，英語が話せないと今日の食事を取るのにも苦労するという状況にはならない．差し迫り方が違うわけなのだ．諸般の事情でアメリカに渡れない場合には，こうした状況に匹敵する条件を自ら作りだし，そこへ自分を追い込むことが成功の秘訣のようである．たとえば洋画を字幕や吹きかえなしに楽しみたい，カラオケで英語の歌を歌ってみたい，ブロンドの美人と仲よくなりたい……．最後のがいちばん強力なような気もするが，こういった動機によって学ばずにはいられないという状況に身を置くことが英会話習得への早道と思う．これは英会話に限らず，何か未知のものを身につけようとするときには，こういった動機（不純なものになりがちだという気もするが，考えてみれば不純ではない動機のほうが少ないのではないだろうか）があると心強いものである．

さて，そこでコンピュータの話である．せっかくコンピュータをもっているのだから，プログラミングというものを習得してみたい，そう思ったらぜひこの「動機」を得るべきである．通常ならば，動機があるから習得する必要が出てくるというものだろうが，現実としてそのような動機がなく，プログラムをしてみたいという状態であれば，後から動機を作る秘訣を身につけようではないか．

プログラム開発用語集
インラインアセンブラ　MS-CやTC++のもつソースファイル中に直接挿入したアセンブラコードを解釈し，コンパイルする機能．

1-7 プロがプログラムを書く原動力

ここでプログラムの作成を職業としている人々のことを考えてみよう．彼らがプログラムを作成し続けられる原動力はどこからくるのだろうか．プロのプログラマの多くは，新人時代になにがしかの「研修・教育」を受けて，システム設計やコンピュータの動作原理などを学んでいるものと思う．また，コンピュータのプログラムが好きでたまらなく，寝食を忘れて勉強にはげむというプログラマの鑑も実在する．そういった教育や努力，才能そして「食うために働かなくてはならない」という現実などが大きな要素になっていることはまちがいないが，1つ忘れてはならないことがある．それは，「大半のプログラマは，与えられたネタを元に仕事をする」ということである．ほとんどのプロは顧客なり上司なりの要求をもとに仕様を決め，実現に向かって作業をしているはずである．

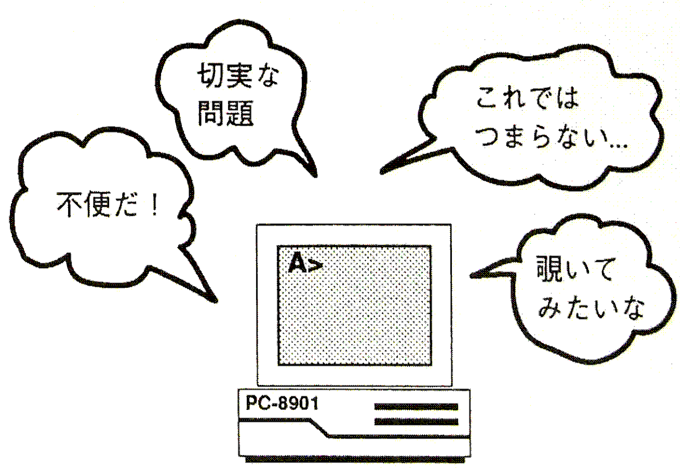
もちろん，まったく何の制約もなく自由な発想のおもむくままに仕事としてプログラムを作るという立場のプロもいないことはないが，一見パラダイスにも思えるこの立場は実はとても苦しく困難なものだろう．制約のない自由な発想というのは，常人にとっては難しいものである．行き先も決めない気ままな旅行にあこがれる人は多いが，実際にそれを実行する人は少ない．パック旅行が人気を集めているわけである．0から何かを考えだす，そしてそれを実行に移すという作業にはたいへんな才能と労力と根気が必要なのである．

1-8 正しいネタの見つけ方

プログラムを書いてみようと思いたってしまったとき，何を書くかのネタがない場合にはたいへん苦労をするという話になってきた．普通は「こういう機能が欲しいからそのプログラムを書こう」という順番になるものを，「何かプログラムを書いてみよう．さて，何にしようかな」の発想なのだから，これはしかたのないことである．運よく誰かから，このようなプログラムを書いてくださいと依頼があればネタの件は解決なのだが，職業プログラマの看板でもあげていない限り，プログラムをあまり書いたことのない人間に作成を依頼する人もいないだろう．

自力でネタを見つけるにはどうしたらよいのだろうか．いちばん早くて確実なのは，言語の入門書などを買ってきて「例題」のようなものに取り組むことである．これなら正解も載っているだろうし，次々と本を買えばいろいろな問題に出会うことができる．ただし，難点もないわけではない．例題はさまざまな読者層を想定して作られているため，必ずしも自分のレベルに合ってはいない．また，例題によって作ったプログラムは自分で欲しかった機能をもったものではないので，でき上がったら便利に使うというわけにはいかず，あまりおもしろくはないかもし

図4　ネタの見つけ方4本柱



れない．ゆくゆくは職業プログラマになろうとして真剣になっているのならまだしも，趣味で作るプログラムがあまりおもしろくないというのは深刻な問題である．やはり，できる限り自分のやりたいこと・欲しい機能を自分で見つけて作っていくのが正しい姿勢であろう．

では，具体的なネタ探しをはじめよう．とはいっても，ただ闇雲にネタネタといいながら歩き回っていても，いいネタにめぐりあえる可能性は小さい．はじめのうちはいちばん効率のよさそうな場所に網をはらなければなかなか獲物はかからないだろう．そこで図4に「ネタ探し4本柱」をまとめてみた．コンピュータを使うときにつねに頭の隅にこの4つのパターンを入れておいていただきたい．問題意識をもち，それを持続させることこそプログラミングへの道の第一歩なのだ．

本題に入る前に，もう1つお節介な話をしておこう．これからネタを自力で見つけていく方法をいくつかご紹介していくわけだが，これらを実践に移す前にぜひとも念頭に置いていただきたいことがある．

①ネタを探す時点では，できるかできないかはひとまず棚上げする．
（とてもできそうにもないことでも，後から検討してみて別の角度から実現可能なこともありうる）

②せっかく摑みかけたネタは，忘れないようにメモをする．昨日見た素晴らしい夢と同じで，あとで思い出そうとしてもきれいさっぱり忘れてしまっている上ネタなどという悲劇にならないように…….

・せっぱつまった問題

人間，困ったときや進退極まったときには自分でも意外なほどの力が出るものである．パソコンを使っていても，「これがないと非常に困る」という問題が発生する．そうなれば誰しもがその問題に対処するための多少の困難や手間は厭わないだろう．

たとえば，ちょっと家人には見せられない性質のプログラムや電子メールのファイルなどをハードディスクに秘蔵していて，もしも留守中に起動されたり閲覧されたりしたらどうしよう……と心配しているとしたら……．鍵が付いていて，ロックすると電源も入らないようになっているパソコンであれば心配することはないが，不幸にしてそういう機種ではない場合には，まさか外出するときにパソコンをかついで行くわけにもいかないだろうから，必死になって方策を考えるだろう．パソコンが立ち上がるときに，パスワードを要求してきて，正しく入力しないと立ち上がらないようなプログラムを入れておくなどというのも有効かもしれない．

また，家人から「パソコンは費用ばかりかかって実用にはなっていないのではないか」という鋭い指摘を受けた場合に，失地を回復するために何か家人をあっといわせるような実用品を作らなければ，今度のボーナスでのメモリ増設計画も風前の灯という事態であれば，家計簿のソフトを作ろう，ついては家人と仕様を検討してということになる可能性もある．

今，パソコンを使う上で非常に困っていることはないだろうか．困っていること，逆に考えればプログラミングの道への大いなるチャンスなのである．

・不便だと思ったら

パソコンが万能の機械だという幻想をいだいている読者はさすがにいないだろうが，実際にこの機械とつき合いだすといろいろと不満が出てくるはずである．これもプログラミングの大きなチャンスだ．不便を我慢しないで「何とかならないか」と考える姿勢を身につけることが，ネタ探し名人への近道である．

寸暇を惜しんで資料に目を通したいときには食事中にでもファイルを閲覧したい．ところが左手には茶碗，右手は箸でふさがってしまっている．いくら空いているからといって，キーボードを足で叩くのは行儀が悪すぎる．こんなときに「次のページ」と声で命令したらそのとおりに画面が変わるファイル閲覧ツールがあったらどんなに便利だろうと思わないだろうか．また，たくさんの文書ファイルがあって，それぞれの最後の行に知りたい情報が書かれているといった状況で，いちいちエディタやワープロを起動してファイルを1つ読み込み，「文末」コマンドを実行するという事態は「不便である」という認識をもつべきである．これを我慢してしまえば進歩がないのだ．少なくとも，「ファイルの最後のほうだけを次々と表示する」という機能があれば劇的に操作性がよくなると考えなくてはならない．

不便だとちょっとでも感じたら，「どういう機能があればそう感じなくなるか」を考える習慣をつけること，これが大切である．

・これではつまらないと思ったら

パーソナルコンピュータを文字どおりパーソナルに使っている場合，その環境は誰からも指図を受けない自分だけのものである．どんな変わった構成になっていようと，常人には理解できないコマンド体系になっていようとかまわない．自分がよければそれでよい

プログラム開発用語集
エスケープシーケンス　ソースプログラム作成時，文字列制御文字を記述するために使われる，0x1b(ESC)や0x5c('¥')に続く表記方法

という世界が構築できる（会社のパソコンをそう考えてはいけない．本体を好みの色に塗り変えたりしたら普通は叱責を受けるであろう）．自分がオーナーであり，極論すれば創造主ですらあるのだから，徹底的に楽しむべきである．

趣味のプログラミングは，この考えの中核をなすものだといってもよい．世界中で誰も使っていない自分だけのオリジナルシステム．プログラムを作ることによって，これが現実となっていくのだ．身近なものを例に取ってみると，ほとんどのユーザーが使うであろうエディタやワープロを起動する前に画面に原稿用紙のような「升目」を表示させておく．その「升目」を消してしまうことなくエディタやワープロを起動する工夫ができたら，もう世界でただ1つの文書入力環境のでき上がりである．

もう1つ考えてみよう．パソコンを使う時間帯のパターンが比較的決まっている人なら，電源を入れたときに自動的にパソコンが時刻を読みとって，朝ならネットワークのアクセスプログラムを起動し，昼休みの時間帯ならゲームを起動し，退社時間間際なら報告書を書くためのワープロを起動するという環境はどうだろう．この機能を広げていけば，たとえば曜日によってプログラムが変わったり，月末だけは経理ソフトが立ち上がるということも可能である．これぞ世界で1つ，自分のための環境ではなかろうか．

・覗き見趣味の奨励

自分が使っているプログラムの中身を覗いてみたいと思ったことはないだろうか．プログラムそのものを解析してというような本格的な興味をもたないまでも，どのようなメッセージ文字列が中に入っているのか知りたくはなかろうか．説明書にも載っていないような恐ろしいエラーメッセージがあったりするかもしれない．中には茶目っ気のあるメーカーもあって，プログラムとはまるで関係のない落書きが入っているものもある．このような興味（悪くいえば覗き趣味）も趣味プログラマにとっては重要なエネルギー源である．

使い方が難しくて苦労しているアプリケーションを，今度は自分が中を覗いてみるというのも楽しいではないか．「コマンドまたはファイル名が違います」などと冷たいことをいってくれるコンピュータシステム自体も，1個のプログラムとしてメモリ上で動作している．プログラムを覗いてみることができれば，システムがもっているさまざまなメッセージも見ることができるだろう．今までに1度も目にしたことがないメッセージを発見して，それがどんなときに出るのかを想像してみるのもおもしろいのではなかろうか．

1-9 選んだネタのさばき方

いろいろなアプローチでネタを仕入れる視点を紹介してきたが，いくらネタばかり集めても，実際にプログラムという行動に移さなくては何もならない．次に考えることは，いかにして集めたネタを現実化していくかということである．

漠然と考えていることがそのままプログラムになってしまうほど現代のコンピュータは進化していない．人間の要求をコンピュータに受け入れさせるためには，より完成度の高い人間のほうから歩み寄っていかなければならない．プログラムというのは，コンピュータに対する命令・依頼の手続きを記述したものであるということは先ほども書いたが，その命令・依頼をふりかえるところから現実化が始まる．これは相手が人間でも同じことではあるが，理不尽なことをいったり，つじつまが合わない要求を出したりすると拒絶されることになる．相手が人間であれば我慢してくれたりまちがいを指摘してくれたりもするが，コンピュータはいわれたことをいわれたとおりにしか実行しないので，つじつまが合わないまま走りだして転んだりする．

ネタの現実化の検討は，要求がプログラミング可能かの判断を意味する．集めたネタを元にそれがどういうプログラムになるかを想像してみよう．そこでまったく見当もつかないようなら，そのネタは料理に適さないかもしれない．また，プログラムとして動く様子が手にとるように具体的に想像できたとしても，現在自分がもっていないオプション機器を必要としていたり，経験豊かなプログラマが50人のチームを組んでも1年もかかる代物だったという場合もある．現在所有していないハードウェアを必要とするというのはわかりやすい判断材料であるが，「自分の手に負えるか」の判断は難かしい．とうてい無理のように思える機能でも，実はちょっと調べると意外に簡単なこともある．逆に，構想段階では楽そうだったプログラムが製作が進むにつれ，たいへんな難物であったということもあろう．だが，MS-DOSを自分ひとりで作ってみようなどという誰が考えても簡単にはいかないテーマを思いつい

プログラム開発用語集
エミュレータ　数値演算コプロセッサと同じ処理をソフトウェアによって行う浮動小数点演算パッケージ．

てしまった場合を除き，最初のうちはここであきらめないで先に進んでいただきたい．ここでおりてしまっては，いつまでもプログラムを書きたい層にとどまることになってしまう．

では，先ほどネタの仕入れ方のところで紹介した具体例を使って，現実化の検討プロセスを見ていくことにしよう．

・パスワードを入れないと起動しないプログラム

キーボードからの入力を受け取って，それをあらかじめ設定してあるパスワードと照合する．合致していればプログラムを終了し，まちがっていれば再びキーボード入力待ちとなる．正しいパスワードが入力されるまで，いつまでもプログラムを終らなければいいわけだ．このプログラムは，キーボードからの入力をどうやって受け取るかと，その入力データを正しいパスワードと照合する部分を作成するだけである．大いに実現の可能性があると判断してよいだろう．

・家計簿ソフト

まずは家計簿とは何かを知る必要がある．どんなことを記入してどんな処理を行うのか．金額や比率の計算なども当然出てくるであろう．プログラムの第一歩は，これから行おうとする処理が何なのかを徹底的にクリアにすることである．これはプロのエンジニアでも同じことで，日本語の文法を知らずして日本語入力プロセッサは作成できないだろうし，麻雀のルールを知らないでマージャンゲームは作れないだろう．

家計簿なら，どんな項目を必要として，どんな計算をするかを実際の家計簿から調べる．日々の支出・収入を数値で入力して，それを月末に集計するといったシンプルな機能だと比較的実現性は高いが，項目名を入れたり，月ごとの変化をグラフに出すといった凝った処理をはじめると大ごとになってくる．最初から何もかも実現しようと思わずに，基本的に求められる機能は何かを分析して取り組めば，実現性も出てくるだろう．

・声で命令できるファイル閲覧プログラム

まず，音声を何らかの形でパソコンに取り込む．その音声データをどんな言葉でも命令として実行するというのは現実問題として不可能であるので，あらかじめ想定している命令語のどれかと合致しているかどうかを判断することになろう．この判断ができてしまえば，あとは通常キーボードからコマンドを受け付けるのとさしたる違いはない．ただし，自分の使っているパソコンに肝心の音声入力機能が備わっていなければ実現は不可能である．目の前のパソコンを見て，マイクらしきものが付いていなければ，現時点ではこのプログラムは作成できないといえる．

・ファイルの最後を表示するプログラム

指定されたファイルを読み込んで表示するという基本的なプログラム手順がわかれば，あとは簡単にことは進む．ファイルを読み込む処理を調べていくうちに，ファイルの任意の位置を読むという機能があることに気がつくだろう．その任意の位置がこのプログラムの場合は，ファイルの末尾である．

一般にファイルの閲覧や内容の操作などは，使用頻度も高いうえ，たいていのプログラミング言語で支援機能をもっていてくれるため，自作のプログラムの入門用として適したジャンルである．ファイル操作関係は，市販品やフリーソフトウェアにも優れた作品が数多いが，自分のためだけのごく私的な機能のプログラムを自由に作成できるようになると楽しいのではなかろうか．

・時刻や曜日によって起動するプログラムを自動選択する

この機能を実現する際に必要となるのは，時刻や曜日をプログラムで取得する方法，その結果によって起動すべきプログラムを選択する処理，そして決定したプログラムを実行させる機能の3つである．こうやって処理をパーツ化して考えていけば，個々の実現方法が見えてくる．

時計やカレンダを画面に表示するプログラムを見たことがあると思うが，パソコンのプログラムで時刻や曜日を取得するのはポピュラーな処理であり，ちょっと調べれば資料はすぐにみつかるだろう．また，プログラムの中から別のプログラムを実行する処理も「メニューソフト」という存在が示すように不可能なことではない．実はいちばんやっかいなのは，取得した曜日や時刻によってプログラム名を決定する処理であろう．ちょっと起動するものを変えたいと思ったとき，容易に起動するプログラムを変更できるような工夫も必要であるし，最初のプログラムはシンプルな機能にして，使い勝手のほうは我慢したほうがいいかもしれない．機能を満載しているが動いていないプログラムと現在のところ不満はあるものの動いているプログラムのどちらが便利かという問題だからである．

作って動かしてみて，「ネタの仕入れ方4本柱」の観点からそのプログラムを評価していく．買ってきたソフトウ

プログラム開発用語集
オーバーヘッド　演算などが実行されない時間のこと．C言語では関数呼び出しは変数参照よりオーバーヘッドが大きい．

ェアと違って，自分の作ったプログラムは，好きなように改造も追加もできるのだから，最初からあせってすべてを盛り込もうとすることもないだろう．

・方眼紙を表示するプログラム

画面上の任意の位置に直線を描く方法がわかれば，あとはどんな方眼を描くかのデザインを決定するだけである．そして，普段使っているエディタなりワープロなりが，前もって描いた方眼を消すことなく起動する方法があるかどうかを調べる．この方法を見つけることがめんどうだからといって，エディタやワープロを作ってしまおうとか，透明なシートに方眼を書いてディスプレィにはりつけるなどということを考えだすと本来のテーマから逸脱してしまうので注意して欲しい．

・プログラムの中身を覗くプログラム

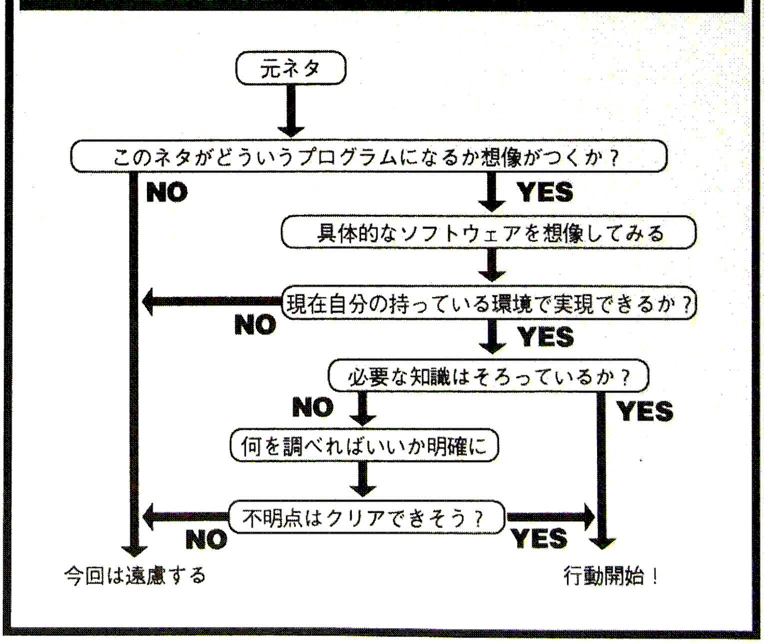
パソコンのカタログを見ると「640KBのユーザーズメモリ標準搭載」などと書いてある．すべてのプログラムはこのメモリ上に呼びだされて動作する訳だが，ほとんどの場合，終了した後に使ったメモリの後始末をしていない．つまり，メモリ上にはそのプログラムの動いた形跡がそのまま残っている状態になっている．これを覗くプログラムを作ると仮定しよう．このプログラムを作るためには，メモリというものをどうやって読むのか，そしてそれをどうやって表示するのかを調べることになる．このあたりは，コンピュータのごく基本的な知識であるので，いわゆる「コンピュータ入門」のような資料をあたれば必ず書いてあるはずである．ただし，コンピュータといっても機種によって微妙にハードウェアが異なっているので，自分の使っている機種のメモリ構成を知ることも必要である．

表示のほうも考えてみよう．よくいわれるように，コンピュータの扱うデータは「０」と「１」の２種類である．メモリの中を覗いてみて，それをそのまま表示させたとしても，人間にはまったく理解できない数字の羅列になることだろう．こんなものを見ても暗記の練習くらいにしかならないので，人間にわかるような形，たとえばメッセージが入っている部分だけを抜き出して表示するといった機能決めが必要になろう．

いくつかのネタを使って，実現化への検討の実例をご覧いただいた．図５にその手順を図式化してみたが，このようにして仕入れたネタを１つ１つ吟味していき，そのネタが実際のプログラムとして作成可能かどうかを選別する．有望なネタがあれば，プログラムとして動作する場合の機能をパーツ化していき，個々の機能がほんとうにソフトウェアだけで実現できるかどうか，できるとしたら何を調査しなければならないのかを詰めていく．特別なハードウェアの追加も必要なく，プログラム機能として理不尽なものでなければ十分に実現可能だろう．

この時点では，できそうな気はするが実際にどうやって作ったらいいのかわからないという微妙なものもあるだろうが，心配することはない．かなり経験をつんだプログラマでも，プログラムを作成するほとんどの場合にいろいろな調べものをするものである．まったく資料をあたらないでスラスラと書き上げてしまうことのほうが極めてまれであるといっても過言ではない．ある問題を処理するプログラムを書こうとするとき，最初は「何がわからないのかわからない」状態である．おそらくプログラムを書いたことがない人や書こうと思って挫折した人がプログラミングに対して感じているものはこの段階であろう．そして，ここから先に進むためにまずしなければならないのは，「何がわからないのか」をはっきりさせていくことである．いきなり考え込んだりしないで，何を調べればい

図５　ネタのさばき方の手順



いのかをリストアップしていくのである．このときに細かい疑問に捕われて悩んだりすると全体像がぼやけてきて，プログラミングそのものに対する意欲がうすらいでしまうものだ．論理的に物事を考えていくというのは，杓子定規に片端から取りかかっていくのではなく，はっきりとした全体像を把握した上で，１つ１つのパーツを確実につみ上げていくことであると認識されたい．

何度もいうようだが，プログラミングとはコンピュータと人間とのコミュニケーションである．茶飲み話のような軽いおしゃべりを楽しむのも，切実なお願いを全霊を込めて伝えようとするのも，まずは自分の気もちをきちんと整理することから始まるものである．そして第２段階として，それを表現するための伝達手段を身につけていこう．これについては次章で詳しく述べることにする．

次界への舟

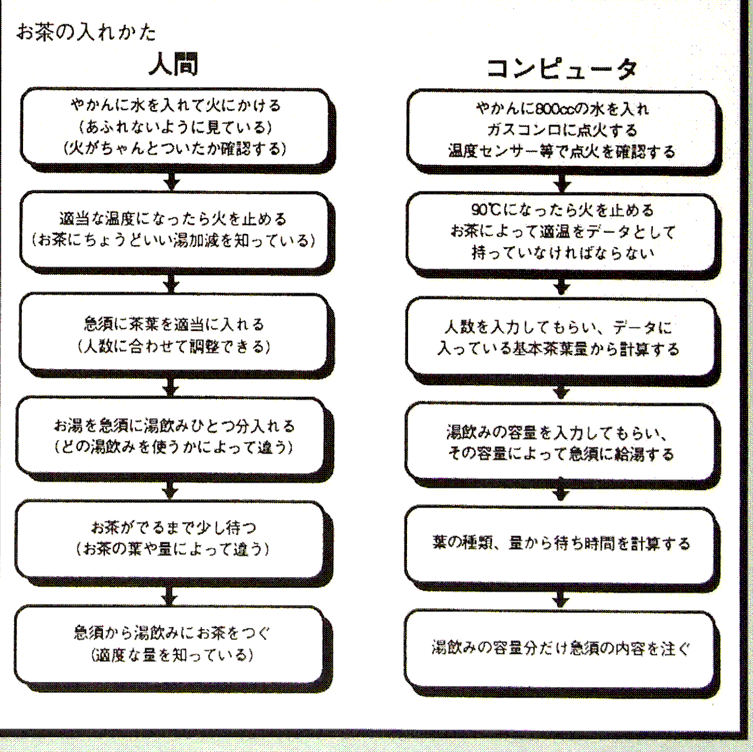
機械界の予備知識

プログラムとはコンピュータに与える依頼事項を整理したものであるという話を進めてきた．プログラムを書くためには，コンピュータに何をさせるかを決めなければならない．「何か書いてみよう」というだけではプログラミングは先に進まないのだ．何をさせるかはっきりさせるために，日頃からコンピュータを使ううえでの疑問や不満をそのままにしないでネタをストックし，そのネタを実現可能なものか吟味する習慣をつけようということであった．

さて，プログラムを書くテーマが見つかり，それが実現可能なものだということになれば，いよいよ次のステップは実際にプログラムの作成段階になっていく．自分のやりたいことがクリアになっていれば，あとは相手に対する「もののいい方」を考えていく．これがプログラミングでいう「コーディング」の段階である．現在のコンピュータは，人間ほど進化していないので，日常的な会話や表現をそのまま解釈することができない．ゆえに人間は「もののいい方」をコンピュータにわかるような表現方法にしてやらなければならないのである．この表現方法をプログラミングというわけだ．

図６では，「お茶を入れる」という処理をする場合の人間とコンピュータの思考形態の違いの一例をあげてみた．これを見てわかるように，コンピュータの思考は「このくらい」とか「適量」などという司令を受けつけない．次の「機械界」では，こういった人間の思考形態をいかにコンピュータの思考形態に変換していくかの手順を紹介しながら，論理の表現方法や必要な知識の調べ方などをまとめてみることにしよう．次の章からの話題として，ネタの見つけ方で出てきた「プログラムの中身を覗く」プログラムというのを取り上げていく．コンピュータの内部事情や表示のメカニズムなどを通じて，「プログラム設計」の現場を体験していただきたい．

図６　コンピュータと人間の意識の違い



プログラム開発用語集
オフセット　ある基準からの相対的な位置のこと、メモリやファイルの位置を示すのに使われる．

2. 機械界の巻

2-1 コンピュータに向かう前に

いままで読者は，「ネタは決まった，さぁプログラミングだ」と走りはじめてどこかで必ずコケる，ということを繰り返してきたのではないだろうか．そして，「今度こそ」の決意も堅くこの特集に目を通してくださっているに違いない．

いったいなにが悪かったのだろう（「頭だ」などとまぜっかえさないように）．プログラミングの入門書はいやというほど読んだし，書いてあることもおおむね理解できた．なるほどこうすればいいのかというのはわかったが，いざ自分でなにか書いてみようと思うとうまくいかない．こう書けばいいというのはわかるが，どうすればこう書けるのかがわからない．

落とし穴があったのだ．「ネタ＝プログラム」ではないのだ．ネタと実際に動くプログラムの間には，暗くて深くて長～い河が流れている．この河をそのままじゃぶじゃぶと渡ろうとするからコケるのである．腕力や体力に自信があればそれも可能だが，普通は河を渡るには橋をかける．橋をかけないまでも木の枝を入れて深さを測ったり，浅そうなところを探したりする．

プログラム作成でこれに相当するのが，「設計」と呼ばれている作業である．

市販のプログラム集や参考書のたぐいは，ページの都合かどうか知らないがこの部分をばっさりと削ってしまっているものが多い．あたかもネタが決まると自動的にプログラムが浮かんでくるかのような書き方だ．ちょっと落ち着いて考えてみればすぐにわかることだが，そんなことあるはずがない．橋なくして河が渡れないのと同様，設計なくしてネタはプログラムにはならない……ならなくはないが，無理矢理ばしゃばしゃと河を渡るとズボンはぐしゃぐしゃ，靴は片方脱げている．プログラムもそんなものになってしまうのが落ちである．

コンピュータにこんなことをさせよう，じゃぁ，え～と，うん，こんなふうにしよう．この「じゃぁ，え～と，」の部分をばっさりと削って，「こんなことをさせよう，うん，こんなふうにしよう」と，ネタからいきなりコンピュータのやり方に飛んでしまっているものが，この手の参考書には実に多い．これじゃ読んだほうはなんのことだかわからない．この「じゃぁ，え～と，」の部分がプログラムの作成でいちばん肝心の「設計」と呼ばれるところなのである．肝心のところがなければわからないのは当然だ．

設計とは，人間がもっとも得意とする「考える」作業なのである．わかってみればなんのことはない．ネタをコンピュータのやり方につなげる肝心の部分は，その方法を考えるという作業であったのだ．むずかしいことではない．いつもしていることだ．

ではこの部分に触れた参考書はまったくないのかというとそうではない．しかし，今度はほとんどの参考書が意地悪しているのかと思うくらいにむずかしい書き方をしている．どうしてこんなことになってしまうのか筆者にはまったく見当がつかない．こういうことを書くとなにか都合の悪いことでもあるのだろうか．

2-2 人間とコンピュータの違い

ネタは人間が考えるもの，プログラ

プログラム開発用語集
オブジェクト指向　オブジェクトに着目し，カプセル化・継承・多態性などによって各要素を抽象化するプログラミング手法

ムはコンピュータが実行するものである．この間に橋をかけるには，両者の違いについて知っておく必要がある．この2つはまったく違うものなのだ．違うものだから間に河があり，橋をかけなければならないのだ．この違いをきちんと理解しておかないと，いつまでたっても橋はかからず河は渡れない．

コンピュータをひとことでいってしまうと，非常に単純な作業を猛烈な速度で繰り返すことのできる機械ということになる．コンピュータを動かしている機構はとても単純なもので，それ自体ではほとんどものの役には立たない．内部の状態はたったのふたとおり．「ここに電気が来ているか，来ていないか」それだけである．これではしかたがないのでこの状態をいくつかひとまとめにして扱うことで幅をもたせているのだが，原理はこれだけなのである．考えるなどという洒落たことはとてもじゃないができそうもない構造だ．

コンピュータは電線がつながっているとおりにしか動かない．ある部分に電気が来ていればそのように，来ていなければまたそれなりの動きしかできない．適当に動くとか，あとでよく考えてみるなどという芸当はそもそも要求するほうが無理なのである．ここをとりあげてコンピュータのことを「偉大なる石頭」などと呼ぶこともある．あとで述べるがコンピュータを構成している半導体は「石」と呼ばれることもあるから，「石頭」とはいい得て妙である．

これに対して人間のほうはずいぶんと複雑である．こちらのほうは考えるという作業ができる．最近はこの部分が欠落しているようにみえる人種も多くなってきたが，それはそれでなにか別のことを考えているのであろう．先ほどコンピュータのすることには適当とかあとでなどということはないと書いたが，人間のほうはほとんどがこれである．人間の思考というのはおおむね「あとで」型であり「適当に」式である．いきなり具体的な手段を並べるのではなく，抽象的なレベルからだんだんに形をはっきりとさせていくというのが普通だ．そして最後にそれは具体的な形をとる．いずれは具体的な形になるのだが，最初からすべてが具体的な形をとっていなければ動けないコンピュータとはかなり様子が違う．

ネタというのは具体的に見えても，落ち着いて検討してみるとかなり抽象的というか，漠然としたものであることが多い．たとえば「プログラムの中を覗く」といっても，どんな形でそれを人間が認識できる形にするのかはまだ決まっていない．それ以前にそもそもメモリとはどういうもので，データはどのように格納されているのかという具体的なものはなに1つはっきりしていないのだ．「あとで」型「適当に」式の人間が考えたのだから，これは当然の話だ．しかしこのあたりのことを1つ残らずクリアにしていかないと，コンピュータに仕事を依頼するわけにはいかない．とにかくこの石頭には，「あとで」とか「適当に」というのが通用しないのだから．

両者の違いがおぼろげながらわかってきた．どうやら一筋縄ではいかないようだが，何事もまず最初に一歩を踏み出すことが肝心．いよいよ「プログラム設計」にとりかかろう．

まずは設計の準備段階，対象物の分析である．対象物がどのようなものであるかがはっきりとわかって，はじめて対応の仕方を考えられるというものである．ここからしばらくはコンピュータの中身の話になるが，考え方はがんじがらめのコンピュータ式ではないから安心してもらいたい．ある程度わかっている人には退屈な話しが長々と続くと思われるだろうが，この長々と続くということ自体に意味があるのだから我慢してもらいたい．一本のプログラムができ上がるのに要する時間のほとんどは設計にかけられるべきだし，その中でもネタの分析はいちばんウェイトを置かなくてはならないところなのだから．

2-3 メモリダンプとはなんぞや

プログラムの内部を覗くプログラムのことを「メモリダンプ」と呼ぶ．では，メモリダンプとは何だろう．強引だが，メモリダンプというからにはメモリをダンプするものである．……これでは説明になっていないではないか．

メモリとは部品屋さんなどで売られている半導体のチップのこと．シリコンから作られるので，他の諸々のこの種の部品同様「石」などと呼ぶこともあるのだが，石だからこのままでは役に立たない．しかるべきところに配線付きで鎮座して，はじめてコンピュータの記憶を司ることができるようになる．そう，メモリとは記憶のための部品である．記憶といっても人間のように忘れたはずのものを思い出したり，覚えているのに忘れたふりをしたりといった器用なものではない．決まった形のデータを決まった形でしか覚えることができず，なにか新しいことを覚えると前のことはすっかり忘れてしま

プログラム開発用語集
オプチマイズ　最適化のこと．高級言語において無駄なコードを省き，プログラムサイズを小さく，または実行速度を速くするためにプログラムを加工すること．

うという情けない代物である．

これではものの役には立たないので人海戦術というか石海戦術というか，大量に用意しておいて次から次からくるデータを，まだなにも覚えていないところに覚えさせていくのである．うまいことを考えるもので，どこいらへんに覚えていて，どこがまだなにも覚えていないのかさえおさえておけば，こんなものでも結構な役に立つ．

ところが，悲しいことにこのメモリというやつ，覚えることはできるのだが忘れることはできない．しかたがないので次にくるデータを，いまなにかを覚えているところに記憶させてしまう．こうすると前の記憶がなくなって新しいデータを記憶するので，結果として前の記憶は忘れたことになる．実に単純極まりないもので，こんなもので動いているのだからコンピュータといってもそう威張ったものではない．

さて，このようにあちこち空いているところに記憶させたり忘れさせたりしていると，そのつどメモリの内容が変わってくる．しかしながらこれがチップの中，石の世界の話なので，目で見てわかるというものではない．これは当たり前だが，それをなんとかして覗いてやろうというのがメモリダンプ．だからといってダンプとは覗きのことであるなんて覚えてもらっては困る．まぁ，ひとかたまりをそこいらへんにざっとぶちまけるというような意味合いなのである．

ぶちまけるといってもいきなりその辺にやられたのでは少々整理に困る．できればコンピュータの中のことはコンピュータの中ですませてもらいたいもので，一般にはコンピュータに備わっている表示装置，通常はディスプレイにぶちまけてもらうことになっている．とはいっても，「はい，メモリをぶちまけて」「はいよ，ザァ～」というわけにはいかない．いかないが，同じコンピュータの中のことであるから折り合いを付けるのはそう難しくもなさそうだ．具体的な方法は後で考えるとして，その前に肝心のメモリの構造を少し勉強しておこう．これがわからずにいきなりぶちまけたのでは，なんのことやらさっぱりわからない．何よりも構造がわからないのではぶちまけようもない．いったいメモリというものは，どういう具合になっているのだろう．

2-4 メモリとはなんぞや

敵を知り，己を知れば百戦危うからずという．ここは，１つ腰をすえて「メモリ」というもののことを知ることにしよう．

まずは図７のような長い長い階段（天国への階段という名曲があった）のようなものを考えていただこう．幅はそれほど広くないが，先は見えないくらいに遠い．ふり返ってみても霞んでしまって後ろも見えない．ここでどこか一段を選んでそこになにか書きつける．これが記憶に相当する．さて，次になにかを記憶させるためには，どこかなにも書かれていない段を探さなくてはならない．いま書いた段にまた書き込んだのではそれまでの記憶がなくなってしまうので，たとえばすぐ後ろ（どちらが後ろかわからないが）の段に書き込んだとしよう．

この２つの記憶の位置関係はこのまま変わらない．１つ１つの段の位置のことを業界の慣習に従って「アドレス」と呼ぶことにする．この場合，記憶した２つの情報のアドレス関係は固定されているということができる．階段の端がどこにあるのかがわからないので絶対位置はわからないのだが，相対位置はわかる．これが相対アドレスである．あとから記憶したほうは，最初に記憶しているアドレスから相対位置にして＋１のところにある．逆にいうと，最初に記憶しているほうは，あとから記憶したアドレスから相対位置にして-1のところにあるということができる．

相対位置では心許ないので，最初に記憶していた位置からステップを数えながら前に前に移動してみると，３段目でステップがなくなった．なるほどここがはじまりであったか．ということは最初に記憶していた位置は先頭のステップからの相対位置にして＋３のところにあったことになる．ここで，先頭の位置からの相対アドレスのことを絶対アドレスと呼ぶことにする．したがって最初に記憶していたのは絶対アドレス３のところである．絶対アドレスは必ず先頭の位置から＋の相対位置なので，わざわざ＋は付けない．

メモリとはこのように，長く長く続

図７　メモリの構造概念

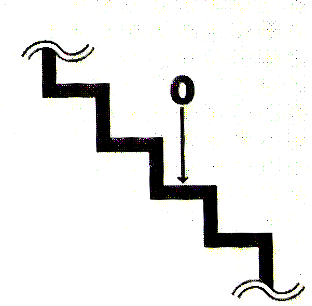
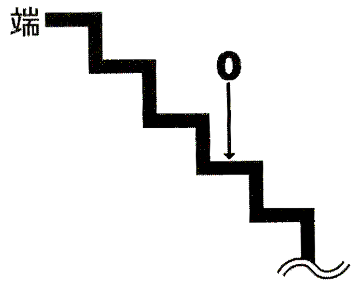


図8　メモリに何か記憶させる



いた階段のように連続したものである．「メモリ空間」などと表現することがあるので二次元的な広がりをもったものかと思われるかもしれないが，実際はこのようなものである．

メモリはどの位置を見ても必ず二種類の位置情報，すなわち相対アドレスと絶対アドレスをもち，同時にその位置固有の記憶をもつ．まだなにも記憶されていないところはどうなるかというと，誰が教えたのでもない，なにかでたらめなものを記憶しているのだ．こんなことでもまったく支障はなくて，そこになにか意味のあるものが書かれているかどうかは，それを使う人間のほうが覚えていればいいことなのである．参照されなければ，なにが書かれていようとそれはないのと同じである．

2-5 記憶されているものはなにか

このような構造のメモリに，どんなものがどんなふうに記憶されているのだろうか．実のところコンピュータでは「いろんなものがいろんなふうに」なのであるが，これではお話にならない．記憶されているものは大きく２つに分けることができる．

１つはプログラム，もう１つはデータである．ちょっと待った，いまこれからメモリダンプのプログラムを作ろうというのだぞ．それなのにどうしてメモリの中にプログラムがあるのだ．けしからんじゃないか．では先にその疑問にお答えしよう．

知っている人には当たり前の話だが，コンピュータというものはメモリだけでできているものではない．メモリというやつは非常におとなしい性格で，教えられるまではなにもせず，教えられたらそれを無批判に覚えこむだけの受動的なものである．決して自分からなにかの行動を起こそうという気にはならないものなのだ．ところがこれだけでは平和ではあるが発展がない．やはり能動的に動き廻るやつがいてはじめてメモリに働く場所が与えられ，世界が活気づくというものである．この能動的な役割を演じるのがCPUという暴れん坊だ．こいつは文字どおりの暴れん坊で，とにかくひとときもじっとしていない．なにかすることはないかとメモリの中を探し回り，そこに記憶されていることをまちがっていようとなんだろうと片っ端から自分流の解釈で実行してしまう．これでは困ってしまうのでCPUを作った人が（といってもテレビによく出てくる半導体工場にいる全身白づくめのお姉さんではない），「目覚めたら最初に命令を読みだすアドレス」というのを決めておいてくれた．いかな暴れん坊といえどもこれには逆らうことができない．メモリのこのアドレスからCPUに対する指示を書いておけば，おかしな脱線をせずにその指示を忠実に実行してくれることになっている．

ところでいったいなにを指示しているのかというと，まず準備体操にあたる各種周辺機器や諸々のICの初期設定，暴れずにちゃんと動いているぞというディスプレイへの表示，そして最後がいちばん大事な，「なにか指示があるまで待ちつづける」という動作の指示である．普通の人間ならばなにもすることがなければぶらぶらしながら指示を待っているかもしれないが，なにしろ能動的なCPUのこと，「なにか指示があるまで待ちつづけることをずっと続けろ」といっておかないとなにをしでかすかわからないのである．まったく厄介なしろものである．

話がそれてしまったが，このようなことのためにメモリにはあらかじめプログラムが書き込まれている領域がある．これは電源が入ってCPUが目を覚ます前に用意されていなければならないので特別な構造のメモリや仕掛けが考えられているが，ここでは深くは追求しないことにする．メモリの内容のもう１つはデータである．ここでデータというのは，CPUが処理する材料，あるいは処理した結果である．

たとえば「２と３を加える」という処理があったとする．「加える」はCPU

プログラム開発用語集
カーソル　マウスなどのポインティングデバイスの指す位置を示すシンボル．

が行う処理であるが，「2」や「3」は処理される材料，すなわちデータである．加えた結果の5という数値も，通常はデータとしてメモリのどこかに記憶される．文字やその集まりである文字列も同じようにデータである……といいたいところだが，コンピュータは文字や文字列は扱えない．コンピュータが扱えるのは数値だけ，それも非情に限定された範囲の数値だけなのである．文字はどうした，図形はどこだと探してみても，メモリの中にはどこにも見当たらない．敵は文字やら図形やら数値やらなんでもかんでも覚えこんだふりをしているが，実は内部では数値としてしか認識していなかったのだ．それも限定された範囲．くわしい理屈は後回しにするが，いま覗いているメモリでは，あるアドレスに記憶できる数値の範囲は0から255の間という取り決めがあるようだ．ずいぶん半端だが，なにぶん相手の都合なので致し方ない．この範囲の数値しか記憶できないというのだから，こんなところに文字だの図形だのが入り込む余地はない．文字は文字だし図形は図形だ．数値ではない．いったいどうなっているのだ．

2-6 文字はどうした図形はどこだ

これではコンピュータでワープロやお絵かきソフトが動いているのが理屈に合わない．理屈に合わないと落ち着かないので，このからくりを探ってみよう．数値しか記憶できないメモリに数値以外のものを記憶させるためにはどういうからくりが必要だろう．

まず文字だ．これに通し番号を付けてしまう．アルファベットは26文字，小文字を加えてこの倍の52文字になる．数字を入れても10文字増えるだけだ．若干の記号を追加してもなんとか255に収まる．なんとも乱暴な話だが，こういう取り決めをしてしまう．つまりメモリに記憶されているのは相変わらず数値なのだが，これを文字として解釈することができるように文字と数値の間にこちらで勝手に関連付けをしてしまおうというのである．たとえば10という数値は'A'という文字をあらわし，'1'という文字は20という数値であらわされるという決め方である．なんとも安易で便利な取り決めだが，これで文字を記憶することができるようになった．もっともこれではひとりひとりがそれぞれ勝手に対応表を作らなければならないので不便この上ない．あるひとは'A'を3という数値だと決めているし，別の人にとって3という数値は'Z'のことである……というようなことではお互い話が通じない．そこで権威筋が集まって一族の内で共通の対応表を作ってしまった．パソコンのマニュアルの最後のほうのページに載っていることがあるので見かけたことがあるかもしれないが，図9のようなものである．

0から255まで，全部で256個の升目に文字が並んでいる．ずいぶん空きが目立つが，空いているところにもその国やコンピュータの事情でいろいろな文字が関連付けられている．この対応表は実は一種類ではないのだが，これが広く用いられている．

どうやらこれで文字を記憶できることになった．文字自体を記憶しているわけではないが，数値と文字の対応が

図9　コンピュータの文字表

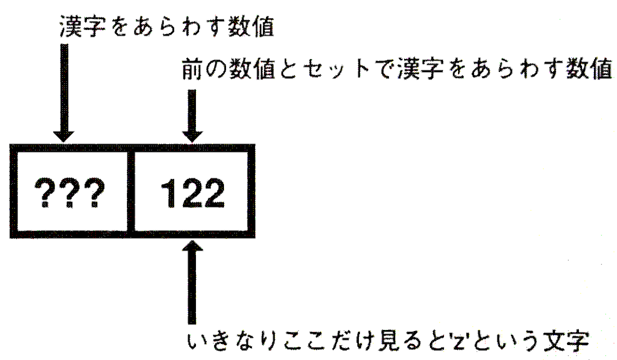
			0	@	P		p								
		!	1	A	Q	a	q								
		"	2	B	R	b	r								
		#	3	C	S	c	s								
		$	4	D	T	d	t								
		%	5	E	U	e	u								
		&	6	F	V	f	v								
		'	7	G	W	g	w								
		(	8	H	X	h	x								
		)	9	I	Y	i	y								
		*	:	J	Z	j	z								
		+	;	K	[	k	{								
		,	<	L	¥	l										
		−	=	M	]	m	}								
		.	>	N	^	n	‾								
		/	?	O	_	o									

決まっていれば，文字を数値として記憶し，記憶した数値を文字として解釈することは可能である．ところで漢字はどうしたらいいのだろう．漢字はアルファベットや数字と違ってとにかく数が多い．全部覚えているわけではないので自慢にはならないが，何千という数なのでとても256種類の中に収めることはできそうもない．それもかなり使われてしまって，もうあまり空きがない．さあどうするか．

やはり世の中頭のいい人というのはいるもので，「1つで足りなきゃ2つ使え」ということになった．256+256だから512…違う違う，256＊256で，なんと6万を超える数値を扱うことができるので，いかに種類の多い漢字といっても不足はない．もっとも，つなげるといっても物理的につなげて漢字記憶専用メモリに仕立て上げたわけではない．あるアドレスと相対アドレス＋1の内容をひとまとめにして扱うということに決めたというだけだから，あるアドレスの数値が漢字の片割れなのか単独の文字なのかはわからない．そこで，さきほどの表の中で空いている升目の中からある範囲を割り当て，その範囲の数値だったらこれに続く次の数値は必ず漢字の片割れという取り決めをしている．図10でその表現方法の一例を見ていただきたい．うまいことを考えるものだがよく見ると落し穴があって，先頭から見ていけばすんなりと漢字だということがわかるが，適当なところをひょいと覗いたのでは，はたしてこれが漢字の片割れかどうかはわからない．漢字処理の盲点となりやすいところであるので，頭の隅にでもとどめておかれたい．

さて，若干の不安を残しつつもこれで漢字を記憶することもできるようになった．ここで先の例にならって漢字

図10　漢字の表現の一例



の表を書くといきなりページが増えてめでたいのだが，編集部からどんな仕返しがくるかわからないのでやらないことにする．

次に図形はどのように記憶すればいいだろう．これには万国共通の取り決めというようなものはないが，大きく分けると2つ．1つはある大きさの空間を考えて，開始点と終了点とか，中心と半径というように数値であらわせる情報をそのまま記憶する方法．数値が大きくなったときは漢字の例にならって2つでも3つでもひとまとめにすれば，理屈の上からはいくらでも大きな数値を記憶することができる．もちろん図形記憶専用メモリを作ったわけではないので，何個ひとまとめで1つの数値をあらわしているかは人間が決めること．メモリの知ったことではない，というのも漢字の例と同じである．

もう1つは図形を点の集まりと見て，ある範囲のどこに点があり，どこにないのかを直接数値として記憶してしまうという方法である．これには1と0という2つの状態しかもたない2進数を応用するのがうまい方法である．2進数というのは0に1をたすと1，1に1をたすと2ではなくて，桁上りして10（「じゅう」ではなく，「いちぜろ」と読む）になるという体系だが，これに「点があれば1，なければ0」という規則をあてはめると，点の集まりを数値として扱うことができるようになる．たとえば点1つおきの点線は，1010101010……とあらわすことができ，直線は11111111……という2進数として扱えばいい．これもまた，数値が大きくなればメモリをいくらでもひとまとめにして扱えばいい．

このように，メモリというものはそれ自体はある限られた範囲の数値を記憶するものに過ぎないのである．それをどのように扱うか，解釈するかでさまざまな情報を記憶することが見かけ上できるということなのだ．

ところで先に，メモリに記憶されているものは1つはプログラム，もう1つはデータであると書いた．しかしここまできておわかりのように，メモリに記憶できるものは単なる数値であって，文字だの図形だのといった区別は人間がつけるものである．プログラムもまたこの例外ではない．つまり，あるアドレスに記憶されている数値をCPUに対する特定の指示命令として扱うのである．この対応をつけるのはこんどは人間ではなくCPUである．もちろんCPUとて人間が作り出したものだからそういう意味では人間が対応をつけているのではあるが，ここは

プログラム開発用語集
記憶クラス　変数をメモリエリアに保持する期間や，結合状態のこと．宣言の構文，またはその構文が置かれた位置によって確立される．

素直に，「CPUが数値を読み込んでそれを命令として解釈している」と考えておこう．

こういうことだからなにかのまちがいで，たとえば文字が記憶してあるところをCPUが命令として読み込んでしまうという事件が起こり得る．もちろん通常はこういうことは起きないが，人間があらかじめ決めておいたCPUに対する指示のかたまり（これがプログラムである）をうっかりまちがえて，CPUがおかしなところから命令を読みだすようになってしまってもCPU自身にはそれをまちがいだと見抜く機能はない．文字列を，あるいは図形を，もしくは大小さまざまな数値を命令として解釈してひたすら実行するだけなのである．これが世にいう暴走である．

2-7 再びメモリダンプとはなんぞや

メモリダンプとはなんぞやを追いかけてコンピュータの奥深くまで探検してきたが，どうやらなにをしなければならないのかがはっきりとわかってきたようだ．メモリとは連続した領域にそれ自体は意味をもたない数値を貯えているもので，それに意味を与えるのは人間の仕事であること．プログラムさえも実は数値の集まりであり，それがCPUに与えられたときはじめてCPUにとっての特定の意味をもつものであること．そういうことがわかってきた．

その数値をどのように解釈するかは人間の自由であり，文字や文字列として解釈してもいいし，ある範囲の数値の連続として解釈することも許される．もちろんCPUへの指示として解釈してもいいのである．メモリにはそのような区分はないので，あるアドレスに必ず決まった用途のデータやプログラムがあるとわかっている場合以外は，人間のほうで解釈のしかたを考えてやらなければならないのである．そして，そうであってもなお，それを別のものとして解釈をすることは自由なのである．

したがって，メモリをダンプするといってもそれを数値として解釈するのか文字として扱うのか，数値にしてもどんな取り決めの数値なのかをこちらが決めてやらなければならない．何度もいうようだがメモリ自身にはそのような区別はないのだから，人間が「ここにはプログラムがある」とか，「この範囲を文字として解釈する」と決めてやる必要があるのだ．

だから，メモリをダンプするにもいろいろなやり方がある．想定したデータの形式に合わせて，いちばん見やすいようなダンプのしかたを考えなくてはならないのだ．

それでは，いままでの話をまとめて，そろそろ次の章に乗り込むことにしよう．

メモリダンプとは，あるアドレス範囲のメモリの内容をあらかじめ人間が決めた使い方に合わせて固有の形態で見せることである．この範囲にあるものがアドレス2つ分ずつまとまった数値であれば2つをひとまとめにして数値として見せなければならないし，文字であれば，個々の数値を文字として，もし漢字も含まれているのならばそれを考慮して見せなければならない．無理やり1アドレスずつ単なる数値として見せてしまって人間のほうで適当に解釈するという乱暴なやり方もあるが，これでは意味のあるダンプにはならないのである．

次界への舟

泥沼界の予備知識

我々は前章で，メモリをダンプするにあたって必要なメモリの構造に関する知識を得た．メモリとは限定された範囲の数値を記憶する素子が連続したものであり，そこに記憶されている数値はそれ自体なんの意味ももたないということがわかった．それに意味をもたせるのは人間の役目であり，メモリをダンプするときはそれに合わせた形式をとらなければならないこともわかった．

ここで話しは少々飛躍する．読者は「逆アセンブル」という言葉を聞いたことがおありかと思うが，これも広義の意味ではメモリダンプの一種と呼んで差し支えなかろうと筆者は考えるのである．CPUに対する命令を少しは（あくまでも「少しは」だが）人間が見てわかりやすい形式で表現したアセンブリ言語というものがあり，これをCPUが理解できるような数値の列に変換する作業をアセンブル，この作業を行うシステムをアセンブラと呼んでいる．逆アセンブルとは文字どおりこの逆の

作業を行うもので，アセンブルされた数値の列をアセンブリ言語の形式で見せるものだ．アセンブリ言語とアセンブルされた数値の列には一対一の関係があるので，こういうことが可能になる．

たとえば"NOP"というアセンブリ言語の命令がある．これは"No Operation"，つまり「なにもしない」という意味の命令だが，ある種のアセンブラではこれは0という数値に変換される．変換後の0という数値を見せられてもなんの感動もわかないが，0が"No Operation"というCPU命令に相当する数値であることを知っている逆アセンブラをとおすと，"NOP"（No OPeration = 何もしない）というアセンブリ言語の命令を示してくれる．メモリに記憶されているものは数値であり，それがCPUに読み込まれることによってCPUにとっての特別な意味をもつ．メモリに記憶されている数値を特定の意味をもたせてそれに合った形式で見せるのだから，これもまたメモリダンプの一種と考えてよかろうと思うのである．

さて予告して寄り道をしてしまったが，こうしてメモリの性質がわかり，そこに記憶されているものがどのような意味をもつものかがわかってくると，おぼろげながらメモリダンプがなにをしなければならないのか，どのような構成要素をもつべきなのかが見えてくる．もちろんこれはひとりひとりの設計者の思うままであって，「これが唯一絶対に正しい．ほかはすべてまちがい」というものではない．だから，以下の展開をそのままうのみにするのではなく，これを参考にし，または下敷きにし，または悪例として自分なりの設計を進めていただきたいと思う．

3. 泥沼界の巻

3-1 メモリダンプの本質に迫る

前章で「メモリダンプとはなんぞや」とやったが，わかったことはメモリに記憶されているそれ自体では意味をもたない数値を，解釈のしかたを決めてぶちまけることだという，実にふにゃふにゃした話しか出てこなかった．ここでもう少し話を進めてみよう．なにによらず，お手本というものはあるものだ．ここで一般にメモリダンプと呼ばれているプログラムがどういうふうになっているのかを眺めてみたい．読者の中にはメモリダンプくらい書いたことあるぞという人もいるかもしれないが，それならけっこう，それをちょっと引っ張りだして眺めてもらいたい．

図11はいま筆者の手元にたまたまあったメモリダンププログラムの出力例である．多少の形式の違いはあれ，おおかたのメモリダンププログラムはこのような出力をする．出力先はたいていの場合はディスプレイである．1行目と2行目はガイダンスであり，それ以下の行の先頭にある8桁の数字はその行のダンプ開始アドレスである．

図11　メモリダンププログラムの出力例

```
-address-| 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
------------------------------------------------------------
00000000 | 50 01 0D 1E 36 09 80 FD 8C 07 80 FD 36 09 80 FD
00000010 | 36 09 80 FD 43 77 60 00 08 06 D3 1B 36 09 80 FD
00000020 | 59 1A 80 FD 12 01 6F 18 37 09 80 FD 37 09 80 FD
00000030 | DC 17 80 FD 1E 00 00 DC 37 09 80 FD 36 09 80 FD
```


プログラム開発用語集
疑似命令　コンパイル時などに，目的ファイルへ変換されるのではなく，処理系に対し指令を行う命令をさす．

図12

```
50 01 0D 1E 36 09 80 FD 8C 07 80 FD 36 09 80 FD
36 09 80 FD 43 77 60 00 08 06 D3 1B 36 09 80 FD
59 1A 80 FD 12 01 6F 18 37 09 80 FD 37 09 80 FD
DC 17 80 FD 1E 00 00 DC 37 09 80 FD 36 09 80 FD
```

これらの飾りを取り除くと，図12のようになる．

これがこのメモリダンププログラムの基本的な出力である．連続したメモリ領域から１つずつ記憶された数値を取り出し，それを16進表示というコンピュータ世界ではお馴染みの表示方式で出力している．ガイダンスはなくなったが，数えてみると１行につき16個の数値を出力していることもわかるだろう．これを見る限りメモリダンプというものは，連続したメモリ領域から１つずつ順に数値を取り出し，１行につき16個ずつ16進表示をするものであるということになりそうだ．しかし待てよ，ほんとうにそうなのか．ほんとうにそれがメモリダンプなのだろうか．

連続したメモリ領域から順に数値を取り出さなければならないのだろうか．１つおきではメモリダンプではないのだろうか．必ず１行に16個出力しなければならないのだろうか．８個ではだめなのだろうか．12個ではメモリダンプとは呼ばないのだろうか．出力は16進でなければならないのだろうか．10進ではいけないのだろうか．これはたまたまこうなっているだけなのである．要求がこうだから，あるいはすでにあるものがそうなっているからといって，たまたまそうなっているだけに過ぎないものと必ずそうなっていなければならないものとを混同してはいけないのだ．ここをはっきりとつかんでおかないと，がんじがらめで改良も流用もできない石のようなメモリダンププログラムができ上がってしまう．石をダンプするんだから石のようなプログラムでいいじゃないかと冗談を言っている場合ではない．

3-2 基本はどこだ

ここからメモリダンプにどうしても必要なエッセンスを抜き出してみよう．最低限なにがあればメモリダンプなのか．なにがよけいなのか．なにがメモリダンプの基本なのか．

まず，必ずしも連続した領域から順に数値を取り出してくる必要はない，と断言してしまう．もちろん数値を取り出してこないことにはダンプにならないから，数値を取り出してくるのは不要な処理ではないが，「連続した領域から順に」である必要はなさそうだ．連続した領域に順にデータが収まっている場合にのみ，これは必要な条件になる．

たとえばメモリ上に年齢と身長と体重という順でひとり分のデータが構成されているとしよう．これが何人分もつながって記憶されているところから，年齢のデータだけを出力したいとすると，データは３つおきに取り出される必要がある．必ずしも連続して数値を取り出さなければならないということではないのだ．このような構造で年齢と身長と体重のデータをダンプしなければならない場合にのみ，「連続して」という条件が加味されるのだ．ここは「ある規則に従って次に処理すべき数値を取り出してくる」ということが決まっていればいい．先に分析したように，メモリは必ずしもそういう使い方をされると決まっているわけではないのだから．

１行につき16個の数値という取り決めはどうだろう．これはメモリダンプにどうしても必要な決まりだろうか．べつに８個でも12個でもいいのではないか．いや，そういうことを決める必要はないのではないか．もちろんなんらかのフォーマットは必要だろうが，これはどうも次の16進出力と同じ次元の問題のようだ．16進出力は決まりだろうか．２進でも10進でもいいのではないだろうか．メモリダンプが16進だというのは誰が決めたのだろう．べつになんでも，都合のいい出力があればそれでいいのではないか．ここは先の16個問題と合わせて，「なんらかの形式に従って数値を処理して出力する」という決め方で十分であろう．

さて，かなり強引だがこれでメモリダンプのエッセンスが抽出された．

メモリダンプとは，メモリからある規則に従って次に処理すべき数値を取り出し，それをなんらかの形式に従って処理し，出力することである．

ということである．めでたしめでたし……というわけにはいかない．これでは１つぽつんとダンプして終わってしまう．どれだけダンプするのか，あるいはいつ終わるのかという項目が必要のようだ．

メモリダンプとは，終了条件が成立するまでメモリからある規則に従って次に処理すべき数値を取り出し，それをなんらかの形式に従って処理し，出力することである．

これではどうだ．一見完璧のようだが，「メモリ」というのが気になる．かなり人間の思考に合わせて抽象的になっているが，ここだけ妙に生臭いではないか．メモリダンプなんだからメモリがあって当たり前だが，メモリという限定がなければどうなるだろう．「ダンプ」になるではないか．

ダンプとは，終了条件が成立するまである対象物からある規則に従って次に処理すべき数値を取り出し，それをなんらかの形式に従って処理し，出力することである．

これですっきりした．ここまでくるとなにやら禅問答めいてくるが，これがダンプのエッセンスだ．この「ある対象物」がメモリである場合をメモリダンプと呼べばいいのである．すなわち，「メモリダンプとはメモリをダンプすることである」ということだ．メモリのほかにどんなダンプがあるのだと聞かれるとここではまだ困るのだが，いちばん基本的な部分を抽出する段階であるのでお目こぼしをいただきたい．

3-3 構成をつかもう

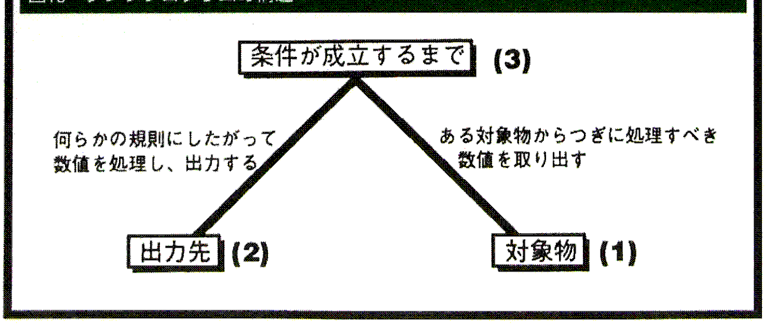
基本はこれでわかった．これをわかりやすいように分割して並べてみよう．

1．ある対象物からある規則に従って次に処理すべき数値を取り出す．
2．なんらかの形式に従って数値を処理し，出力する．
3．1と2を終了条件が成立するまで繰り返す．

メモリダンプは以上の3つの部分からなることがわかった．この1つ1つを「モジュール」という名前で呼ぶことにしよう．

どうやら全体を制御しているのは3番目のモジュールであるようだ．1番目のモジュールは数値を取り出す処理なので，取り出したあとどうするかは教えられていない．2番目のモジュールにしても，数値がきたら処理をするのであって，自分で数値を取りに行く手段は与えられていない．3番目のモジュールが，あらかじめ定められた終了条件が成立するまでこの2つのモジュールを制御しているというのが全体の構成のようだ．これを図式化したものが図13である．

図13　ダンププログラムの構造



実にすっきりとした構造である．1つの能動モジュール（3）が，2つの受動モジュール（1，2）を制御してひとまとまりの動作をする．それぞれのセクションは自分に与えられた単純な処理を実行すればよく，お互いに干渉するところがない．

ところで，ガイド（スケール）表示はどうすればいいだろう．これはいま分解した3つのモジュールのどれにも属していない．なくても支障はないがあると便利というレベルのもので，いわゆる「おかず」だ．おかずは好きなものを気分に合わせて選びたいので，ここでは考えないことにしたい．最初に一回出てくればいいだけなので，あとでなんとなく寂しいと感じたらちょっと付け加えればいいのだ．

さらにアドレス表示はどうしよう．これは出力処理の処理単位ごとに出てくる．これはガイドと違って，ないとかなり不便な気がする．これがなければどこをダンプしているのかわからないのだから．さて，どこに入れるべき処理だろうか．少なくとも数値を取り出すモジュールには馴染まないことがわかる．ここには「出力」という概念がない．だとすると出力モジュールか，これらを制御しているモジュールだが，できれば制御モジュールにもご遠慮願いたいところだ．なぜなら，制御モジュールには出力モジュールの内部のことは知らせないですませたいからだ．それを知ってしまうと出力モジュールを取り替えるときに制御モジュールも変更を余儀なくされる．これではなんのために処理をきれいにモジュール分けしたのかわからない．制御モジュー

プログラム開発用語集
高級言語　CPUの構造に束縛されない言語．アセンブリ言語のようにCPUよりでなく，より自然の言語に近い．

ルがアドレス表示をするということは，とりも直さず出力モジュールの内部の様子を知ることになる．アドレスは数値を取り出すモジュールが知っているのだからそこからもらうことは可能だが，「ここが表示の始まりだ」というタイミングを知らなければアドレス表示はできないからだ．

制御モジュールも候補から外れた．では出力モジュールの中に入れるとしたらどうすればいいだろう．それではこうしよう．出力モジュールに渡すデータを，数値とそれが置かれていたアドレスの2つにする．「アドレス」といっても，必ずしも物理的なメモリのアドレスでなくてもいい．たとえばダンプ開始アドレス（これは物理的なアドレス）からの相対アドレスであってもいいのだ．要はデータとその出所をペアで渡すということだ．それをどのように解釈し，どのように使用する（または使用しない）かは出力モジュールの自由とする．こうしておけば，出力モジュールのほうは当然表示開始のタイミングを知っているのだから，そのときに出所を使えばいい．ふだんは出所は使われないので無駄になるが，たったこれだけの無駄で制御モジュールは出力モジュールのことを知らなくてすみ，出力モジュールはアドレス表示に関してフリーハンドを獲得したことになる．アドレス表示をしたくなければしなくてもいいのだ．

3-4 閑話休題

「ものを考える」話題を集中的に書きつづけてきたので，ここらで一息入れることにしよう．仕事中ではない方は，冷たいビールでもいかがだろうか．

さて，メモリダンプなんか作ったことあるぞという読者は，ここまでしつこく考えてみたことがおありだろうか．「あーメモリダンプなんて簡単簡単，ポインタとバイト数もらってさ，ちょいちょい……」とばかり，リスト1のようなプログラムを書いていたのではないだろうか．

もちろんこれはこれで正しい．たしかにまちがいなくメモリダンププログラムの一例である．それにケチをつけようというつもりはないが，はたしてメモリダンププログラムの本質を理解して書いていたのだろうか．応用はできるだろうか．要求が変わったときにどこを見ればいいかすぐにわかるだろうか．このプログラムを文字列をダンプするように変更してほしい．文字列の区切りは数値の0，これがきたら改行するようにしてくれ……といわれたらどうすればいいか，すぐに答えが出せるだろうか．それともそんなものはメモリダンプではないから新しく別のプログラムを書けばいいとお考えだろうか．そしてそれはたいした手間ではないと．

どんなに簡単にみえることでも，しつこくしつこく考えてみると意外なことがわかってきたり，とんでもないアイディアがわいてきたりすることがあるものだ．それはすぐに目先のことには使えないかもしれないが，いつかは必ずその人のためになる．思いついたことはなんでもいいからノートにでも書き留めておく癖をつけてほしい．できれば時々はそれを読み返してまた考えてみることだ．プログラマにかぎらず，物事をいいかげんに考えていて大成した人間などいないのだ．

3-5 徐々に肉付けをしていこう

いきなりCなどもち出して思わぬ寄り道をしてしまったが，ダンプの骨格はでき上がった．もちろん骨格なのでこのままでは動かない．これに肉を付け形を整えて，少しずつまともなプログラムにしていくことにする．

まず最初に，どのようなダンプをするのかを具体的に決めておかなくてはならない．ここでは先にとり上げた例と同じようなものを作ることにしよう．

リスト1

```
void    DumpMemory(unsigned char *Cptr, int Length) /* いきなりCしてごめんね */
{
int   i, Line;

        Line = (Length + 15) / 16;
        while(Line--)
        {
                printf("%04X ", Cptr);
                for(i = 0; i < 16; i++)
                {
                        if(!Length--)     return;
                        printf("%02X ", *Cptr++);
                }
                putchar('¥n');
        }
}
```

プログラム開発用語集
構造化　プログラムの制御が上位のモジュールから下位のモジュールに渡される階層構造をとるプログラミング手法

すなわち，指定された数だけ連続したメモリ領域から次々に数値を取り出し，それを１行16個ずつアドレスつきで表示するというやり方だ．いちばんボリュームがありそうなのは数値を規則に従って出力する処理のようだ．まずこれから片付けることにしよう．

1）出力モジュール（数値を規則に従って出力する）

これは受動的に動作するモジュールである．データは外部から与えられるものであって，自分から取りにいくものではない．人間社会では必要のないことはできるだけ教えないというのが無用なトラブルを避けるコツであるが，これはプログラムでも同じである．したがってこのモジュールに自分でデータを取りにいく手段を与えてはならない．あくまでもデータはこのモジュールを動かすときに外部から与えるようにすべきである．

そうすればこのモジュールは，そのデータがどこに置かれていたのかを気にせずに働けることになる．プログラマから見ると，どこから取ってきたデータであっても，たとえプログラムの中で作り出したデータであっても，出力のしかたが同じであれば出力の処理にはこれを使うことができる．こういう配慮がプログラムの再利用を容易にするのだ．１度作ってちゃんと動いたものを別のシーンでも使い廻すことができれば，プログラムを作成する時間がその分だけ節約できる．同じものを何度でも書くことが好きだというのであれば致し方ないが，たいていは同じことをするのに時間をかけたくはないものだ．このように，プログラムを設計する段階から再利用を考えていくのがデキるプログラマになるための近道である．

さて，処理の詳細を考えるつもりでまたまた寄り道をしてしまった．「デキるプログラマへの近道」などと柄にもないことを書いてしまったので，少々赤面しながら先へ進むことにしよう．

このモジュールは与えられたデータを一定のフォーマットで出力する．フォーマットはこのモジュールだけが知っており，他から教えられるものでも他に教えるものでもない．フォーマットは自由に決められるが，取り合えずここでは前述の一般的なダンププログラムと同じものを出力することにきめたので，そのセンに添って設計を進めよう．

まずアドレスを表示し，次に16個分のデータを出力するというフォーマットである．アドレスのあとや各データの間には，わかりやすいように空白を置いておくことにしよう．この辺の細かなことはこのモジュールの中だけのことなので，いくらでも自由に決めることができる．独立しているということは自由だということなのだ．

ところでフォーマットは自由だが，このモジュールには制限が１つだけある．出力すべきデータが外部から渡されるということである．ただ１つだが，これは意外に重要なことだ．自分で勝手にデータを取りに行ってもいいのならば，「まず最初にアドレスを出力して，次にデータを１つとってきて……」という具合にできる．しかし外部からデータとともに起動されるという構成になっているので，起こされてはっと気がつくと膝の上にデータが乗っているという具合になる．これを16個分表示したら改行して，またアドレスを表示してから次のデータを……というふうに進むわけだが，このデータが16個のうちの何番目のデータであるかをどうやって知ればいいのだろう．呼び出したほうは教えてくれない．むこうはこちらの内部のことは知らないので，何番目のデータかなどということには頓着しないのだから．したがって自分のほうで入ってきたデータの数を数えておかなくてはならない．これを一般に「カウンタ」と呼んでいる．データのカウンタだから「データカウンタ」だ．これを最初は０にしておき，データがくるたびに（ということは自分自身が呼び出されるたびに）１つずつ進めていく．そうすれば今度きたデータが何番目のものかがわかるというわけだ．

16個表示したところでまたデータカウンタを０にしておけば，次の16個もまた同じようにカウントすることができる．めでたしめでたしであるが，１つ問題がある．先に「最初は０にしておき」とあっさり片付けてしまったが，いちばん最初にデータカウンタを０にするのはだれだろう．出力モジュール自身がこれを前提として動くのだから，自分で設定することは不可能である．どうやらこれは自分を呼び出すモジュール，すなわち制御モジュールに０にしておいてもらう必要がありそうだ．ただし，制御モジュールに「０」という，出力モジュール以外にはなにを意味するのかわからない値を覚えてもらうわけにはいかないので，モジュール内に「このモジュールの内部を初期状態にする」という処理を特別に用意して，これを最初に呼び出してもらうことにしよう．

出力モジュールはいろいろなタイプのものが考えられるが，そのいずれにも「このモジュールの内部を初期状態にする」という処理を用意しておく．こうすれば出力モジュールを取り替えてもその他のモジュールは変更の必要はない．以後，この特別な処理のこと

プログラム開発用語集
最適化　高級言語において無駄なコードを省き，プログラムサイズを小さく，または実行速度を速くするためにプログラムを加工すること．

を「初期化処理」と呼ぶことにしよう。

次に出力処理本体を考えることにする。しなければならない仕事は以下のようなものだ。

【モジュール機能一覧】

A．アドレスの16進出力。
B．データの16進出力。
C．改行。

アドレスは行の最初に出力すればいいのだから，これはカウンタが0かどうかを見ればすむ。改行はアドレスを出力する直前か，データを決まった数だけ表示した直後にすればいい。どちらでも好きなほうを選べばいいのだが，ここではどのような状態から始まっても必ず行の先頭位置からダンプできるように，アドレスを出力する直前ということにしておこう。

これで手順が書ける。出力モジュールの処理手順は以下のようになる。

【モジュール内部処理手順】

1．データが与えられたときにカウンタが0ならば，改行してアドレスを表示する。
2．与えられたデータを16進で出力する。
3．カウンタを1つ増やす。
4．カウンタが16になったら0にする。

最後に，データが16に満たないうちに処理が終わってしまう可能性を考えて，呼び出してもらう処理を用意しておこう。ここで1回改行しておけば，次からの出力に支障をきたさなくてすむ。なにごとも準備と後始末が肝心だ。これをきちんとやってはじめて一人前と世間が認めてくれる。この後始末の処理を「後処理」と呼ぶことにしよう。

これで出力モジュールに関して決めておかなければならないことはすべて解決した。念のために出力モジュールを使用する外部からの手順を書いておこう。

【モジュール使用手順】

1．初期化処理を呼び出す。
2．任意の数だけ，データを与えて呼び出す。
3．後処理を呼び出す。

2）データ取得モジュール（データを規則に従って取り出す）

次は出力モジュールが必要とする数値と出所のペア（以後，これをデータペアと呼ぶことにする）を取得するモジュールである。このモジュールがしなければならないことはただ1つ，数値の取得のみであり，出力モジュールのような複雑な処理は必要ない。ただひたすら，呼び出されるたびにデータペアを返す。数値をどのようなやり方で取得するか，次のデータがいまのデータとどのような関係にあるかを知っているのは，唯一このモジュールのみである。これは出力モジュールがそうであったように，だれから教えられるものでもなく，まただれかに教えるものでもない。

簡単なようだが落し穴があるかもしれない。ここは1つ出力モジュール同様しつこく考えてみることにする。このモジュールは出力モジュールと同じで，受動的に動作する。制御モジュールから「データペアをよこせ」といわれたときはじめて動くのである。勝手にデータを取ってきてどこかに送り込むというようなことはない。出力モジュールとの違いは，データペアをもらうのではなく呼び出し元（制御モジュール）にデータペアを返すという点である。

データ取得モジュールは，どこから数値をもってくるのだろうか。メモリダンプではいうまでもなくメモリからなのだが，メモリのどこからもってくるのだろう。それをデータ取得モジュールは最初から知っているのだろうか。知らなければそれはどこから与えられるべきだろうか。先に書いたようにデータ取得モジュールは，次のデータペアがいまのデータペアとどのような関係にあるかを知っている。「次」がなにを意味するものかは自分自身が知っているのだが，いちばん最初はどうするのだろうか。

これをデータ取得モジュール自身が知っているのがまずいということはすぐにわかるだろう。そんなことになっていたら，このモジュールを使ったメモリダンププログラムはいつも同じアドレスからダンプを行うことになってしまう。これは少々まずい。どうやらこれは制御モジュールから与えられるのが順当のようだ。出力モジュールのときに出てきた初期化処理である。これをこのモジュールを使うときに1回だけ呼び出してもらう。次からは自分自身が「次」の意味を知っているのだから，制御モジュールの手助けはいらない。図らずも初期化にけりがついてしまったので，次は処理の本体を考えることにしよう。何度も書いているがこのモジュールの本体は，呼び出されるたびに次のデータを返すものである。「次」がなにを意味するのかは制御モジュールは知らない。このモジュールだけが知っていることである。

ここではメモリから数値を取り出す

ことに決めているので，数値を取り出すには初期化で制御モジュールから与えられたアドレスを使用する．アドレスはメモリ上のある一点をポイントしているので，ここからもってくればいい．このようにある場所をポイントしているものをポインタと呼んでいる．数値は連続して取り出すことになっているから，数値を取り出した後でポインタは次を指しておかなくてはならない．ここでいう次とは，現在のアドレス＋1のことである．このように1つ進めることを「インクリメントする」などと表現する．ここで取り出した数値と，ポインタの値を返せばいいのである．ポインタの値はインクリメントする前のものが必要なのでまちがえないようにしなければならない．後処理はなにか必要だろうか．今回のモジュールではとくになさそうだが，出力モジュールに合わせて一応入口だけは用意しておくことにしよう．

以上を出力モジュールにならってまとめると以下のようになる．

【モジュール機能一覧】

A．数値とアドレスを返す．

【モジュール内部処理手順】

1．ポインタが指すメモリ上の数値を取得する．
2．ポインタを1つ進める．
3．数値とポインタ（1つ進める前のもの）を返す．

【モジュール使用手順】

1．アドレス付きで初期化処理を呼び出す．
2．任意の数だけ，呼び出してデータを取得する．
3．後処理を呼び出す．

3）制御モジュール（条件が成立するまでデータ取得と出力を繰り返す）

いよいよ最後の「制御モジュール」である．これが全体の動作を決定付けている重要なモジュールだ．重要なわりにはこのモジュールがしなければならないことは単純である．データ取得モジュールと出力モジュールを，ある条件が成立するまで繰り返し呼び出せばいいのである．いちばん簡単な条件は，「ある数だけダンプしたら」というものである．しかし，さきほど両モジュールで棚上げにした問題をここで解決しなければならない．それぞれのモジュールの初期化と後処理の呼び出しである．

出力モジュールの初期化処理は，ただ呼び出せばいいだけである．これに対してデータ取得モジュールの初期化には，取得開始アドレスを与えてやる必要がある．呼び出す順序に制限はない．各モジュールが独立していて，依存関係が非常に希薄だからである．モジュールの独立性が高いと，使用するほうで気をつかわなければならないことが少なくなる．こうなるようにモジュールを設計することがデキるプログラマの……おっと，また出てしまった．

その初期化だが，各モジュールの初期化はどういうタイミングで行うべきだろうか．これはもういうまでもないと思うが，制御モジュールの初期化の中である．制御モジュールはこのダンププログラムを代表しており，外から見えるダンププログラムとはすなわちこの制御モジュールなのだ．この中でのみ意味がある出力モジュールやデータ取得モジュールのことは外部に知らせる必要はない．ダンププログラムを使うときにいちいち出力モジュールの初期化だのデータ取得モジュールの後処理だのとやっていたのではめんどうでかなわない．後処理はこのモジュール自体には必要ないが，出力モジュールの後処理はやはりこの代表モジュールの後処理の中で呼び出さなければならないし，データ取得モジュールにもここではなにもしないが，後処理がある．

処理の本体だが，とりあえず決められた回数だけ繰り返したということを終了条件にしてみよう．このモジュールの初期化で回数を与えてもらい，その数を記憶しておいてデータの取得と出力を1回呼び出すたびに1つ減らす．これが0になるまで繰り返せば期待した結果が得られるのである．出力モジュールでつかったものと同じで，これもまたカウンタだ．ただしこちらは，インクリメントではなくデクリメント（1つ減らす）である．

ところで，出力モジュールとデータ取得モジュールについては初期化は本体の処理とは別になっている必要があった．これらは最初に1回だけ呼び出され，本体の処理はいつ呼び出されるか，何回呼び出されるかわからない状態で待機していなければならないからだ．しかし制御モジュールに関しては，初期化処理，本体処理，後処理が1回ずつこの順序で実行されることがはっきりしているので，この順序で処理をならべてしまうことができる．外部から見ると1つの処理だけが見えることになるので使い勝手もいいから，このような構成にしてしまおう．

さて，ここまでの内容をまとめてみ

プログラム開発用語集
シフトJIS　JIS漢字コードをシフトし，グラフィック文字部に割り当てたコード体系．MS-DOSや一部のUNIXで採用されている．

よう．

【モジュール機能一覧】

A．メモリをダンプする．

【モジュール内部処理手順】

1．内部で使用する各モジュールの初期化処理を呼び出す．
2．与えられた繰り返し回数を記憶する．
3．繰り返し回数だけデータ取得モジュールと出力モジュールの呼び出しを繰り返す．
4．内部で使用した各モジュールの後処理を呼び出す．

【モジュール使用手順】

1．アドレスと繰り返し回数で呼び出す．

3-6 かくして設計は終わった

メモリダンプ1つになんでこんなにまでとお思いだろう．白状するが，筆者もそう思う．設計というものの重要性とその手法の一端をご紹介したいためにここまでしつこくやってみたのだが，あまりにエネルギーを必要とするのでプログラムを作るのがいやになってしまった読者もいるかもしれない．

しかし，ちょっと待ってもらいたい．ものを作るというのはエネルギーを必要とするものではなかったか．プラモデルでもいい，料理でもいい，彫刻でもイベントでもなんでもいいが，なにか作ったことがあるひとにはわかっていただけると思う．ものを作り出すというのはたいへんなエネルギーを必要とするのだ．どうしてプログラムだけが例外であり得ようか．この労力を厭う人には，ものを作り出すことは永遠にできない．この労力が楽しいと思う人でなければものを作り出す人にはなれないのだから，楽しいと思わない人はプログラマになることは早々にあきらめたほうがいい．楽しいと思うひとと，ここまできたのだからやめるのは悔しいと思う人だけ，最後のステージに進もう．

3-7 プログラミングとはコンピュータとの会話

設計が終わって，なにをどうしなければならないのかはすべてはっきりした．人間ならばこのまま渡せばちゃんとメモリダンプをしてくれることだろう．もちろんメモリの中が読める人間がいたらの話ではあるが．

ところが相手はコンピュータである．このコンピュータというしろもの，そう簡単に動いてはくれない．なにしろ人間ではないのだ（あたりまえだ）．人間ならばそれこそ「はい，このとおりにメモリダンプ1つ」ですむこともあるが，コンピュータにはそもそも人間の言葉が通じない．もっとも，最近では言葉が通じるはずの人間でも「これコピー，B4でね」といわれて地下4階をうろうろしていたOLという話も聞くので油断はできない．かように意志の疎通とはむずかしいものだ．言葉が通じるはずの人間同士でさえこうなのだから，相手が言葉の通じないコンピュータとなると一筋縄ではいかない．むこうが人間の言葉を理解してくれないのと同じように，こちらもコンピュータの言葉などとんと馴染みがないのだから当然といえば当然の話だ．しかしこれでは話しが先に進まないので，なんとか折り合いをつけなければならない．こういう場合，昔は事情がわからないほうが腰を低くして教えを乞うたものだが，賢いほうがにこにこしながら歩み寄るというのが最近の風潮のようだ．世間の風潮とあらば致し方なし，コンピュータの言葉をこちらが理解してやることにしよう．

3-8 コンピュータの言葉は意外に簡単

言葉は内面を伝えるものである．伝える内面が複雑であれば，言葉もそれに耐えるだけの複雑さをもたなくてはならない．「ヤッダー」「ウッソー」「ワカンナーイ」くらいで伝わってしまう人種がいるようだが，それだけ内面が単純なのである．コンピュータもこれほどひどくはないがかなり単純な構造なので，言葉も単純だし語彙も少ない．

したがって１つ１つの単語や基本的な文法を覚えるのはそうむずかしくない．むずかしいのはその組み合わせだ．英会話を例にとると，英語を母国語としている人たちは我々が中学校で学んだ程度のじつに簡単な単語を上手に組み合わせて豊かな表現をしている．この組み合わせのしかたが理解できるまではかなりたいへんだが，いったんわかってしまうと後はぺらぺら……と，買ったきりほとんど読んでいない英会話本の表紙に書いてある．組み合わせ方というのはなかなか教えにくいものだ．要はその場その場で最適な組み合わせを考えていくということに尽きるのだが，こういう芸当をこなすのは頭がよくなくては無理だ．もっとも頭がよければそんなことは教えなくても自分で身につけていくし，そうでない人に百万ケース並べても，たった１つの教えていないケースで立ち往生してしまう．結局教えてあげられるのは基本だけ．後は本人の努力次第というところなのだ．

……と少々脅かしておいて本題に入るが，幸いなことにプログラミングには英会話と違っていくつか有利な点がある．英会話のように人前で声を出さなくてはならないものはまちがえると恥ずかしいし，うまく伝わりそうに思えても「この表現は幼稚かな」などと考えてしまって次が出てこないなどという情けないことがある．しかも普通の生活をしていると英語を話す機会はほんとうに少ない．しかしプログラミングとなると組み合わせをまちがえても，最悪の場合でもコンピュータがだんまりになるくらいで実害はない．コンピュータを破壊するプログラムなどというのは，かなり腕のたつプログラマが気合を入れて書かなければならないので，入門したばかりの人は安心してよい．しかもプログラミングというのはたいていひとりでやるものだから，人に見られているというプレッシャーもない．たとえ表現は少々幼稚でも，でき上がったプログラムがちゃんと動いていれば誉められることこそあれ恥をかくことはない．さらにコンピュータはかなり安くなっているので，自分用に１つ買ってしまえばいつでも好きなときに練習することができる．したがって才能の部分は致し方ないとしても，「本人の努力次第」の部分を頑張れる条件が整っているのである．

3-9 コンピュータにほんとうにできること

銀行のCD機から現金を払い出したり，その現金をゲーム機でまた取り上げたり，コンピュータというのは実にさまざまな悪さを……いや，働きをするものである．これだけいろいろなことをするのだからさぞかし複雑な構造になっているのだろうと思うとあにはからんや，これが実に単純なのである．CPUによって多少の違いはあるが，コンピュータにほんとうにできることはせいぜい加減乗除と比較くらいのものである．基本的にはCPUは最後に読みだした命令のすぐ後から次の命令を読みだすが，さすがにこれでは役に立たないので，次に命令を読みだすアドレスを変更する機能だけがお情けでついている．これを利用して，ある条件が成立した（あるいは成立しない）ときに次の命令を読みだすアドレスを変更するように命令しておけば，「判断」という高級な働きを行うことができる．

3-10 高級言語を使う

CPUにできることは実はこの程度のことなのである．したがってCPUに語りかける言葉も実に単純なものだ．しかしこれではあまり単純すぎてかえって使いにくいので，少しは人間のほうに近い（それだけCPUにとっては難しい）言葉が考えられている．いわゆる高級言語とか高水準言語と呼ばれているものがそれで，多くの言語設計者が使いやすいようにいろいろと工夫をしている．

代表的なものとしてはC，Pascal，FORTRAN，COBOL，BASICなどがある．パソコンでの主流はなんといってもCだろうが，最近ではよりよいCとしてC++などというものも使われはじめている．ここにあげた言語は，それぞれ特徴はあるものの基本的な構造は非常に似通っている．コンピュータの言葉をより人間に近づけようとして考えられたのだから，構造が似通ってくるのは当然であるが，おかげで１つの言語を理解すればこの系統のほかの言語も比較的理解しやすいということになっている．この，代表的な言語が非常に似通った基本構造をもっているということは，どの言語を使うかはプログラムを作成する上で大きな問題ではないということである．どれかの言語でいったんプログラムを書き上げ

プログラム開発用語集
条件コンパイル　ヘッダ指令を使用して，コンパイルする部分や，しない部分を作ること．

図14

出力モジュール

(モジュール機能一覧)

A. アドレスの16進出力。
B. 数値の１６進出力。
C. 改行。

(モジュール内部処理手順)

1. データが与えられたときにカウンタが０ならば、改行してアドレスを表示する。
2. 与えられたデータを16進で出力する。
3. カウンタをひとつ増やす。
4. カウンタが16になったら0にする。

(モジュール使用手順)

1. 初期化処理を呼び出す。
2. 任意の数だけ、データを与えて呼び出す。
3. 後処理を呼び出す。

データ取得モジュール

(モジュール機能一覧)

A. 数値とアドレスを返す。

(モジュール内部処理手順)

1. ポインタが指すメモリ上の数値を取得する。
2. ポインタをひとつ進める。
3. 数値とポインタ（ひとつ進める前のもの）を返す。

(モジュール使用手順)

1. アドレス付きで初期化処理を呼び出す。
2. 任意の数だけ、呼び出してデータを取得する。
3. 後処理を呼び出す。

制御モジュール

(モジュール機能一覧)

A. メモリをダンプする。

(モジュール内部処理手順)

1. 内部で使用する各モジュールの初期化処理を呼び出す。
2. 与えられた繰り返し回数を記憶する。
3. 繰り返し回数だけデータ取得モジュールと出力モジュールの呼び出しを繰り返す。
4. 内部で使用した各モジュールの後処理を呼び出す。

(モジュール使用手順)

1. アドレスと繰り返し回数付きで呼び出す。

てしまえば，必要があればそれを他の言語で書き直すことは比較的容易である．少なくともここに上げた言語であればそういうことができる．核となるのは，論理の表現であって，それをどう「いいまわすか」は，副次的な要素なのである．

では，いままでのことがらに基づいてプログラムの設計を進めていこう．念のために前章までの内容を図14にまとめておく．

ネタの探し方で「プログラムの中身を覗く機能」というのを思いついてから，いろいろと検討を重ね，ここまでやってきた．プログラム設計の現場というのはおおむねこのようなものである（いや，実情は……自粛することにしよう）．こういった作業をしていると，周囲にも「いやぁ，ここのところ設計が行き詰まっててねぇ」とか「ここの仕様がね，もうちょっと考えてみる余地はあるわね」などと，それらしい話題が出せるというものである．

さて，仕様がここまできたら，あとは実際のプログラムコーディングである．プログラムは仕様を守り，そのとおりの内容をこれから使うプログラミング言語のしきたりに合わせて書いていけばでき上がることになっている．ここでは，特定のプログラミング言語は使わずに，仮想的に日本語によるプログラミング言語を想定して書いてみることにしよう．リスト２をご覧いただきたい．

すでにこのプログラムが頭の中でCやPascalのプログラム表現になりつつある読者，「なんでこんなのがプログラムなんだよぉ」といいたげな読者，「どっかで見たことあるぞ，これ」とにやつく読者，さまざまな顔が見えてくるようである．いつの日にかこのようなプログラミングでサッと動作するコンピュータシステムが日本での常識になることを夢見ている昨今である．

さて，頭の中ではダンププログラムが動きだし，16進文字列で埋め尽くされてきたことと思う．ここで最後にもう一度「プログラムのモジュール化」

リスト２　仮想日本語プログラムによるメモリダンププログラム

```
出力数は１２８。

「データ出力の初期化処理」は
        データ出力カウンタを0にすること。

「データ出力後処理」は
        なにもしない。

「データ出力」は
        データ出力カウンタが0ならば改行してポインタを出力して空白を出力して
        数値を１６進で出力して
        データ出力カウンタを1つ増加させて
        データ出力カウンタが１６以上ならデータ出力カウンタを0にすること。

「データ取得の初期化処理」は
        アドレスをポインタにセットすること。

「データ取得の後処理」は
        なにもしない。

「データ取得」は
        ポインタが指すところから数値を取り出し
        ポインタとともにデータペアを作り
        ポインタをひとつ進めて
        データペアを返すこと。

「メモリダンプの初期化処理」は
        アドレスで「データ取得の初期化処理」をして
        「データ出力の初期化処理」をすること。

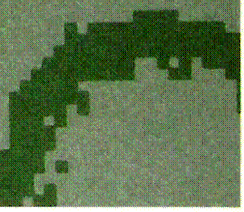
「メモリダンプの後処理」は
        「データ出力の後処理」をして
        「データ取得の後処理」をすること。

「メモリダンプ」は
        「メモリをダンプします。開始アドレスを入力してください」と出力して
        アドレスに開始アドレスを入力して
        アドレスで「メモリダンプの初期化処理」をして
        出力数の回数だけ「データ取得」と「データ出力」を繰り返し
        「メモリダンプの後処理」をすること。
```

という話をして，この特集をしめくくることにしよう．いままで何度も何度も，モジュールの独立性について書いてきた．また，「モジュールを入れ替えたときにも」などと思わせぶりなことも書いた．実際にはどういうご利益があるのか，リスト２に書いた仮想プログラムを例にとって説明しよう．

このプログラムの「データ取得」という部分は現在「ポインタが指すところから数値を取り出し」という記述をしており，データはメモリから取得するようになっている．これをメモリからではなくファイルから取得するように変更すれば,「ファイルダンププログラム」のでき上がりである．また，「データ出力」の部分を逆アセンブラにしてしまえば，ほかは一切変更なしに(といっても逆アセンブラを１本書くのはたいへんな変更だが)，ファイルからデータを読みだして逆アセンブルを行うプログラムに大々変身する．このように処理を適切にモジュール化しておくことでプログラム改造時の手間が大幅に削減できること，再利用が容易になることはプログラミングを続けていくためにたいへん重要なことである．ぜひここのところは肝に銘じてもらいたい．

4. 終りの巻



自分でもあきれるくらいにしつこくやってきたが，多少なりとも読者の参考になっただろうか．プログラムを書くときにはまず対象を分析し，きちんとした設計をしてからコンピュータに向かうべきであること．設計の段階で全体を再利用が可能なように独立したいくつかのモジュールに分解し，いままでに用意しているモジュールはそれを最大限に活かすべきであること．そしてなによりも，プログラムを作るということは考えることであるから，それを厭ってはならないことを，なんとかわかっていただこうと，ゴールデンウィークを丸々つぶしてあがいてきた．ここからなにか１つでも吸収していただければ，休日にどこにも連れていってもらえず隣の部屋でふてくされている原稿書きの妻も少しは浮かばれよう．

プログラム開発用語集
スクロール　ウィンドウ内に表示できない情報を見るために，スクロールバーを使って上下左右に移動させることができる．

特集
過激なプログラミング入門

高級言語でプログラミングできるようになったからといって，安心するのはまだ早い．高級言語は人間の思考に都合のいいように作られているだけに，コンピュータのハードウェアを覆い隠し，見えにくくしてしまう．コンピュータの本体は"石"，つまりCPUである．この"石"を理解するための仮想CPUを設計し，一通りの[illegible]を完成させてみることで，プログラムとハードウェアの結び付きを理解することができるだろう．既存のCPUの命令一覧を眺めるだけで，設計者の意図が読み取れるようになれば一人前である．

アセンブラレベルでの命令を理解するには
CPUを設計するのが早道だ

石からはじめるプログラミング その1

虎岩登

前口上

過激なプログラミング入門です．過激といっても何が過激なのでしょう．こういうお題をもらっても，ちょっと困ってしまいます．そこで，困ったときの広辞苑頼み，引いてみると，「過激――度を越えてはげしいこと．社会的な思想や行動にもいう．」では，ストールマン率いるFSF（フリーソフトウェアファウンデーション）みたいなことなのでしょうか．あれは確かに過激な運動というべきかもしれないけど，私がなにかここで書くべきことではないようです．編集部からのメモには「ハードウェアに近いところで，高級言語の対極になるような」と，あります．そうすると，度を越えてはげしくハードウェアに近いプログラミングについて書くべきなのでしょう．

だいたい，高級言語のよくないところは，コンピュータのハードウェアのイメージを隠しすぎて，目の前にある機械が何を中で行っているかが見えにくくなってしまうことです．多少はアセンブラなどをかじったほうが，プログラムの効率を考えたりしやすくなると思います．高級言語でもなんでも，おなじ振る舞いをするプログラムはかなり自由にいろいろな書き方ができるけど，できる人はだいたい同じような書き方をします．どんなオブジェクトが吐き出されるか，だいたいわかっているから，そんなに効率の悪い書き方をしないように自然とそうなっていくのです．それに，オブジェクトの様子がわかっていたほうが，デバッグの時や，他人の書いたプログラムを調べたりするときにも得をすることがあります．

えっ，高級言語の最適化も高度になっているし，デバッガだってソースレベルでほとんどすべて解決できるって．それに，それなりの機能をもったアプリケーションのように大きなプログラムを作成するときは，細かいところまで考えていられないって．うーん，そうかもしれません．ただ，それでも機械の動きについてイメージは持っていたほうがよいと思うんですけど……．

ということで，ここではアセンブラレベルのプログラミングについて書きます．しかし，アセンブラレベルというと，たいていの人は苦手ということになっています．これにはいくつかの理由があると思います．

ひとつには，入出力のライブラリがなくて，全部自分で作らなければならないこと．これは敷居が高いですよね．だいたいMS-DOSなどの，オペレーティングシステムのサービスルーチンを覚えるだけでたいへんです．Cのライブラリを一通り使えるようになるより面倒くさい．アセンブラ自体には，フォーマット付きの入出力などないから，それを使いたいなら自分で作るか人が作ったものをもらうしかないわけです．

さらに，高級言語ほど便利な表現が用意されていないから，とにかくプログラムの字面が長く平坦に見えてしまう．理解しにくい．またこれは80x86系の悩みですけど，セグメントの取り扱いがわかりにくいということでしょう．もっとも，どうせアセンブラだけでプログラムするなら，普通はtinyモデル（すべてのセグメントレジスタが共通）ですむ程度の大きさにしかならないでしょうから，入門レベルではそんなに問題はないと思います．でも高級言語とインターフェースをとる場合などは，結構面倒ではあります．

そして最大の欠点は，命令がやたらに多くて，全部を把握しきれないってことではないでしょうか．とにかく種類が多く，しかもよくわからない禁止事項や注意書きが多い．これをしたいから，こう命令を組み合わせて……なんて自在に考えることができるようになるのはいつのことになるのでしょう．そんなことを心配したくなるくらい，最近のCPUの命令の構成は複雑になっています．

でも，実際にはそんなに困惑することもありません．それぞれの命令は，それなりの必然性があって，設計されているわけですから．その必然性の元になっている体系が頭の中にできれば，それぞれの命令はその体系の中の然るべき場所に納まるはずです．

そういうわけで，CPUレベルの機械命令が，どうしてそんな風にきめられたか見ていこう．というのが本稿の目的です．特定のCPUの命令の成り立ちを調べるのも，勉強になります．が，ここではオリジナルのCPUを設計するところから入りましょう．プログラミングを学ぶには，他人の作成した優れたプログラムを読むのもよいですが，自分で小規模なプログラムをいくつか作成するのもよい勉強になると思います．それと同じ様に，自分で一からCPUの命令体系を作ってみるのも，アセンブラレベルでの，設計者の意図というものを理解するにはよい手段だと思います．

独自のCPUですから，なにか名前が必用ですね．PP-16とでもしておき

プログラム開発用語集
スクロールバー　ウィンドウの右端や下端に表示される．このバーをマウスで操作して上下左右に画面をスクロールさせることができる．

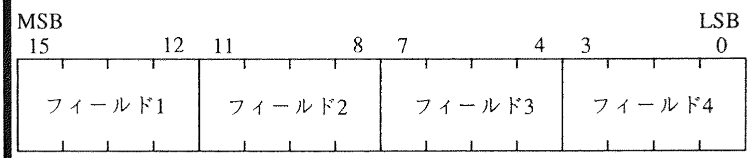
ましょう．

今回はどちらかというと一般論を中心に，独自のCPUを設計していきますが，インテル社の86系列やモトローラ社の68系の資料を用意して比較してみるとよいと思います．ただし，さすがに80386クラスの拡張された32bitのCPUになると，命令体系がかなり複雑になるので，最初は8086の中でも基本的な命令だけを参考にするほうがよいかもしれません．

レベル1 大枠を決める

では，これから設計するCPU，PP-16の大まかな仕様を決めていきましょう．単にコンピュータのCPUの仕様といってもいろいろあります．電源電圧や使用環境などのハードウェア面ももちろんありますが，今回中心になるのは命令体系についてです．基本的な命令長は，レジスタ数はいくつ，どんな演算ができるのか，アドレッシング・モードはどうするか，レジスターレジスタ間演算を基本にするのかレジスターメモリ間演算か，など，それでも多くのことを考えて決めなければいけません．そして本来のCPU設計な

図1　基本的な命令フォーマット(それぞれのフィールドをどういう意味に使うかは命令による)



ら，ここでそれらの多様な組み合わせについて，それなりに細かく検討し，いくつかの組み合わせについてはかなり突っ込んだ詳細設計まで踏み込んで，優劣を決めなければいけないのです．しかし，そうするにはまだ知識もありませんので，まずは天下り式に大まかな部分を決めてしまいます．これから行う技術的決断については，後で反省することにしましょう．

もちろん，基本的な命令の「意図」を理解するためのCPUですから，昔のミニコン風の，できるだけ単純な命令体系を考えることにします．また，段階的に命令を増やしていくことで，最初から複雑な命令体系を扱わないで済むようにします．ですから，後で命令を付け加えやすい命令割り付けをするようにしましょう．

まず命令は基本的に16bitで表されるとしましょう．16bitが1語の16bit CPUです．8bit単位の命令だと，命令長が小さすぎて表現できる命令が少なくなります．可変長の命令体系にすればよいのですが，かえって設計が複雑になってしまいます．16bit長の命令なら，やってみればわかりますが，命令を割り振るのに十分な表現力があります．

さらに，命令の割り付け原則として，16bitを4bitごとの4つの場所に分けます．それぞれをフィールドと呼んで，特別の意味を持たせます．が，この4つのフィールドの意味は，命令を定義する際に明らかになってくると思います．これからの方針として便宜的にここで決めてしまいます（図1）．

命令自身は16bitでおさまるようにしますが，番地指定などでもう1語使う必要も出てくるでしょう．

基本的なデータ長も命令と同じく16bitにしましょう．メモリは，とりあえず16bitアドレッシングが使える65536Byteまで使えることにしておきましょう．今では小さめのアドレス空間ですが，かまわないでしょう．拡張方法を考えるのも，おもしろいかもしれません．当然，バイト単位でのアクセスもできるように設計しましょう．

レジスタ数はいくつにしましょうか．レジスタをいっさい持たないで，すべてメモリーメモリ間演算を行う命令体系もあります．また，レジスタを100以上も持つアーキテクチャのコンピュータも存在します．このあたりはいろいろな考え方がありますが，とりあえず16個の汎用レジスタを用意しておきましょう．名前はR0, R1, ..., R15としておきましょう．そうすると，レジスタ指定が4bitで済みます．4bitというのはさきほど決めた機械命令を構成するフィールドの大きさと一致します．

次に命令の実現の容易さから，取り

表1　PP-16の概念的な仕様

命令長	16bit/1語（場合によっては2語）
データ長	16bit整数，8bit整数
レジスタ数	16個（汎用R0-R15）
レジスタ長	16bit
アドレス空間	16bit（65536Byte）

敢えずレジスターレジスタ間演算を基本とする命令体系を用いましょう．ほとんどのCPUが2アドレス方式を取っていますから，PP-16も同じ方式を使います．2アドレス形式の詳細についてはコラムを参照してください．

ほかに特殊なレジスタとして，プログラム実行時に，これから実行する命令を指しているプログラム・カウンタがCPUの中に含まれています．

レベル2 ロード・ストアと2項演算

では，最初に単純なプログラムについて考えてみます．メモリ内のXという番地とYという番地のそれぞれの内容を加えて，Z番地にしまうだけのプログラムです．C言語で書けば，プログラムというより単純な文で，

```
z = x+y;
```

となるような単純なものです．

このプログラムをPP-16では，どう表現することにしましょうか．レジスターレジスタ間演算を行うことにするので，加算はレジスタ同士になります．ですから，データをメモリ上からレジスタへロードして，加算を行い，結果をメモリにストアすることにします．つまり，

COLUMN

演算対象の指定方法

演算命令では，命令の中に演算対象を指定するフィールドを持つ必要があります．このフィールドの数からCPUのアーキテクチャを分類する方法があります．四則演算のような2項演算には，ALU（算術論理演算ユニット）への入力が2つ，ALUからの結果を格納する場所の指定とあわせて，最大3つの「場所」を表すフィールドが必要になります．実際には，この「場所」というのは，レジスタ指定とかメモリアドレスの指定などになります．

●3アドレス形式

命令の中に，その3か所すべての場所の情報を含むようにしたアーキテクチャです．a = b + cのような演算が1命令で行えることで自由度は高いのですが，なんといっても命令の中に占めるアドレス・フィールドが3つあるために命令長が長くなるのが欠点です．

●2アドレス形式

現在，もっとも一般的に使われる命令形式です．普通はa=a+bのような演算を行うのが基本です．これは，演算結果の格納場所と入力の片方を共通に使うことにして，3アドレス形式のアドレス指定を1つ省略したものと考えることができます．

3アドレス形式でも，a=b+c+dのような計算をする場合は，中間結果を格納する場所を工夫して使う必要があります．たとえば，

```
ADD  A, B, C  ; A = B + Cのつもり
ADD  A, A, D
```

のようになったりします．2番目の命令は2アドレス形式と同じことになってしまいますね．計算式が複雑になったり，とくに配列の添字計算などを行うと，このような中間結果を次々に演算対象にする場面が出てきます．すると，せっかくの3アドレス形式もかなりの割合で，2アドレス形式的な使い方がなされます．

なら，最初からそういう設計法をしたほうが，命令長も短くなって効率が高いだろうというのがこの方式です．

●1アドレス形式

もっと極端な方法がこれです．2項演算の対象を1つしか指定しないとなると，もう一方の対象はどのように指定するのでしょうか．専用のレジスタなどを用意して，演算対象のもう1つはそのレジスタしか使えないもの，と決めてしまうのです．A = A+xのような形で，Aは専用レジスタです．このレジスタは，その命令形式から，加え込む計算（アキュムレーション）を行うのが得意なので，アキュムレータと呼ばれたりします．命令はさらに短く効率的になる可能性がありますが，結果を格納できる場所が1つに決まっているので，中間結果を残したまま別の計算を行う場合に，いちいち別のところに保存する命令を使う必要があります．こんなことは，そんなにないだろうと考えるかもしれませんが，くり返しなにかの計算を行うときに本来の計算の他にループ回数を増やすのにもアキュムレータを使うのですから，意外に足を引っ張ります．

●0アドレス形式

こんなに省略してしまっても平気なのかと思えるような形式ですが，考え方としては意外と身近なものだったりします．FORTHのようなスタック演算を基本にすると，演算対象はスタックの先頭と2番目のデータになりますし，演算結果はスタックにプッシュされるので，別に命令で指定する必要はありません．PascalのPコードなどがこの形式でした．

プログラム開発用語集
セグメント　8086のアドレス空間は1MBある．このアドレスを指定する方法としてSegmentという概念を導入した．

X番地からR0へデータをロード
Y番地からR1へデータをロード
R0=R0+R1
R0からZ番地へストア

という手順になります．これだけのプログラムに，最低限，レジスタ間の加算命令と，データのロードとストア命令が必要になります．

ではロード命令の形式を決めましょう．アセンブラ表記では

```
LD    Rn, Addr
```

とすることにします．機械命令では，図2のように，2語で表すことにします．ここで，フィールド2のRRRRはレジスタ番号で，R0なら0000，R1なら0001，R15なら1111というように，Rnのnの部分を2進数表記したものが入ります．2語目のAAAA AAAA AAAA AAAAは，ロードすべきデータの番地です．

ストア命令も同様に，図3のようにしましょう．ロード命令と異なっているのは，MSB側の4bitであるフィールド1の内容が，0001になっているところです．

PP-16では，このフィールド1を，命令の大まかな動作を決定するためのフィールドに必ず使うことにします．このフィールドを「Iフィールド」と呼ぶことにします．こうすれば，命令を16進数で表示したときに，最初の数字で何を行うべきかがすぐにわかります．どの命令にも必ずIフィールドがあります．

では，次に加算命令です．加算命令も特有の値をIフィールドに割り当ててやればよいわけです．しかし，今は加算しか考えていませんが，一般にはCPUで行いたい演算としては，減算，論理和，論理積，排他的論理和，乗算，除算など，多くの演算があります．しかし，Iフィールドは16通りの値しかとれませんから，それぞれの演算にIフィールドの値を割り振ると命令が表現できなくなってしまう可能性があります．そこで，Iフィールドには2項演算であるという，大まかな意味を割り当てて，Isフィールド（サブIフィールドのつもり）というフィールドを作って，そこに具体的な演算を表すコードを記述しましょう．

つまり，機械命令のフォーマットは図4のようになります．この図の中でレジスタを指定するフィールドについて補足しますと，演算命令はくり返しになりますが，

```
Rd = Rd+Rs
```

という2アドレス方式で行いますから，あるレジスタのデータを別のレジスタの内容に足し込むことになります．一般に，足し込まれるレジスタをディスティネーション・レジスタ，足されるデータを提供するだけのレジスタをソース・レジスタと呼びます．そのディスティネーションの頭文字のdを取ってRdと，またソースの頭文字のsを取ってRsと表記しました．2つのレジスタを指定するための，2つのフィールドが図4のように必要です．アセンブラ表記では，

```
ADD    Rd, Rs
```

と表現することにします．

これで目標のプログラムが書けます．

```
        LD     R0, X
        LD     R1, Y
        ADD    R0, R1
        ST     R0, Z
;
X:      DW     ?
Y:      DW     ?
Z:      DW     ?
```

で，完成です．もっともこれはプログラムの断片なので，このままではST命令を実行した後で変なことになってしまいますので，とりあえず，目標通りメモリ上のデータを操作できたのだからよしとしましょう．

レベル2のまとめ

メモリとレジスタ間のデータ転送命令であるLDとSTを定義しました．これらの命令は絶対番地でメモリを指定します．

2項演算の命令を定義しました．

これらの機械命令のフォーマットは

図2

```
0000 RRRR 0000 0000
AAAA AAAA AAAA AAAA
```

図3

```
0010 RRRR 0000 0000    ST
AAAA AAAA AAAA AAAA
```

図5の通りです．

課題

あなたが使っているCPUのアセンブラ命令について，データ転送を行う命令と，演算を行う命令を抜き出してリストを作ってみましょう．そのときに，PP-16で見たような命令のフィールド割り当ての規則性なども見つかるか，フィールド割り当ての規則にあてはまらない命令が多くてもその規則にあてはまらない命令を利用する必要性は高いかを検討してみましょう．あとから特殊な用途のために追加された命令なのかもしれません．

また，演算命令などで似たようなことを行う命令のフォーマットが何種類かあったとしたら，それはなぜかこの時点で考えてみましょう．

レベル2の注意点

昔は，メモリとレジスタのデータ転送には今回のように別のニーモニックを割り当てることが多くありました．しかし，データ転送という意味では同じことなので，同じニーモニックにしたほうがプログラマにとって単純でわかりやすいという考え方があり，近年のCPUのアセンブラではそうなっていることが多いのです．

具体的には，データ転送にはすべてMOVとかLDというニーモニックを割り当てて，データをメモリのX番地からR0に持ってくるには

```
MOV    R0, X
```

と表記し，逆にメモリへ書き込む場合は，

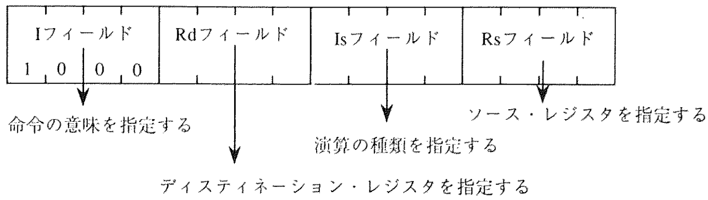
```
MOV    X, R0
```

として，レジスタとメモリアドレスの2つのオペランドを書く位置によって，データ転送の方向を表そうという方法です．この流儀だと，レジスターレジスタ間のデータ転送命令も，

```
MOV    R0, R1
```

となります．

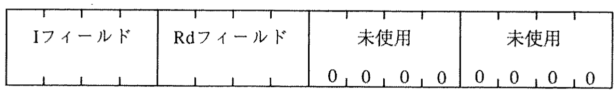
図4　演算命令の定義(加減算だけをとりあえず定義した)



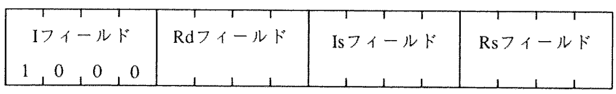
Isフィールドの値	命令	演算内容
0	ADD	Rd = Rd + Rs
1		
2	SUB	Rd = Rd - Rs
3		

Isフィールドの値と演算の対応表
空白は未定義

図5　レベル2で定義した命令



Iフィールドの値	命令	動作
0	LD	メモリからレジスタへデータ転送
2	ST	レジスタからメモリへデータ転送



Isフィールドの値	命令	演算内容
0	ADD	Rd = Rd + Rs
1		
2	SUB	Rd = Rd - Rs
3		

プログラム開発用語集
デバッガ　プログラムを実行しながら，エラーの原因を探す手助けをするプログラム．

レベル3 ループを書く

次に，X番地から連続した10ワードに入っているデータの総和を取るプログラムを書いてみましょう．レベル2で定義した命令だけを使うことにすれば，図6のようなプログラムになりますが，これではあまりに冗長ですよね．それに，11ワード分の総和を取る場合も別のプログラムを書かなければなりません．

では，どのようなプログラムが普通なのかといえば，図7のような「くり返し」を用いたプログラムになるでしょう（この手順には少し問題がありますが）．

このためには，ステップ4の「次の番地の内容をロード」するための命令をどうするかと，ステップ5の「10語分の処理をしたか」の判定法，さらにステップ6の「まだなら，3へ」の命令について考えなければなりません．

まずはステップ6の問題から片付けましょう．通常，プログラムは，アドレスの小さいほうから大きいほうへ並んでいる命令を，その順に実行していきます．が，このように時には実行順序を変更したい場合もあります．そこで一般に用意されている命令がジャンプ命令，またはブランチ命令です．日本語で表記するなら分岐命令ですね．

では，ジャンプ命令を定義しましょう．ジャンプ命令のIフィールドは8を割り当てましょう．アセンブラのニーモニックは，JMPとします．JMPはLD命令と同じく2語を使い，2語目にジャンプするアドレスを必要とします．アセンブラの表記法は，

```
JMP    addr
```

にします．

ところが，このJMP命令は，常にジ

COLUMN

2の補数表示について

2進数で，負の数を表現する方法にはいろいろあります．たとえば，1bitを符号ビットとして用い，残りのビットで数の絶対値を表現することも可能です．しかし，もっとも一般的に用いられているのが2の補数表現です．

負の数の計算が簡単になるように表現方法を決めると，負の数を利用する計算が楽になるはずです．正の数の加算についてはよくわかっているとして，加算が自然に可能になるように負の数を拡張してみましょう．どういうことかといえば，-1という数の表現を定義するのに，

$$1 + (-1) = 0$$

というふうに，符号を反転させた値を加えると0になるという性質が成り立つような負数の表現があれば自然な拡張といえます．

ここで，8bitの数についてちょっと具体的に考えてみましょう．0000 0001という数(今扱っているのは2進数です．4bitごとに空白をいれて見やすくしています）に何を加えても，0にはならないように思えます．しかし，計算結果も8bitしか得られないとすると，1111 1111という数を加えてみるとおもしろいことがわかります．

```
0000 0001 + 1111 1111
= 1 0000 0000
```

と，単純な計算で求まりますが，結果の9bit目は残りません．ですから，答えは0のように見えます．

-2に対応する数は，-1から1を引いた数ですから，

```
1111 1111 - 0000 0001
= 1111 1110
```

と表すことができるでしょう．こうして，任意の負の数が定義できます．でも，8bitだと256通りの表現しかできませんから，正の数と負の数を半々に分けて割り当てます．すると正負の最大の数は127（0111 1111）と-128（1000 0000）となります．こうすれば，最上位ビットが0なら正の数，1なら負の数という性質も成り立ちます．

ところで，ある2進数と，その数をビット毎に反転した数を足し合わせると，常に各桁が1の数（1111 1111）になります．1111 1111 + 1 = 0ですから，任意の数xに対して，

$$x + (\bar{x} + 1) = 0$$

が成り立つので，$-x = \bar{x} + 1$と，簡単な演算で負の数を求めることができます．

プログラム開発用語集

デバッグ　プログラムの誤りを訂正する作業．デバッガを使わず，目でプログラムの流れを追いながらする作業をハンドデバッグという．

ャンプしてしまいます．これでは，ステップ5の「10語分の処理をしたか」のような判定結果にしたがって，ジャンプしたりジャンプしなかったりすることはできません．そこで，フラグというものを導入することにします．フラグとは，日本語なら旗ですよね．宴会芸で両手に赤と白の旗を持って，「赤上げて，白上げないで，赤下げない」なんていう旗を想像するとよいでしょう．この旗はいくつか用意されていて，ALU（算術論理演算ユニット）の演算結果によって，上げられたり下げられたり（セットする，あるいはクリアすると表現します）することになります．現実のCPUでは意外に多くのフラグが用意されていますが，PP-16ではとりあえず2種類のフラグを用意することにします．ゼロフラグとキャリーフラグです．

図6

```
1 レジスタR0に0をロードする
2 レジスタR1にX番地の内容をロード
3 R0 = R0 + R1
4 レジスタR1にX+2番地の内容をロード
5 R0 = R0 + R1
      :
      :        同じことを残りのアドレスについて繰り返す
      :
   終わり
```

図7

```
1 レジスタR0に0をロード
2 レジスタR1に最初の番地の内容をロード
3 R0 = R0 + R1
4 レジスタR1に次の番地の内容をロード
5 10語分の処理をしたか
6   まだなら，3へ
7 終わり
```

この2つのフラグは，演算命令（とりあえずはレベル2で定義した2項演算命令を考えてください）によって，変化します．ゼロフラグは，名前の通り演算結果が0の場合にセットされます．それ以外ではクリアされます．キャリーフラグの「キャリー」という言葉は，足し算をするときの桁上がりのことです．ですから，桁上がりが出た場合にセットされ，それ以外の場合はクリアされます．引き算の場合は桁借りを表すことにします（表2）．

このフラグの状態によって，ジャンプしたり，あるいはそのまま次の番地の命令を実行するか，動作が変わるようなジャンプ命令を作れば，ステップ5の処理が可能になります．条件分岐命令です．さきほど，定義したばかりのJMP命令を，条件を記述できるように拡張しましょう．新たに，フィールド2をCCフィールド，条件を記述するフィールドにして，図9のような命令形式にします．アセンブラ表現は，

```
JMP    cc, addr
```

のように，条件を番地指定の前に書くようにします．無条件にジャンプする命令は，

```
JMP    T, ADDR
```

とも書けますが，

```
JMP    ADDR
```

のように，オペランド部が1つなら，条件のTが省略されたものと解釈すればよいでしょう．

これだけ用意すれば，目的のプログ

図8　JMP命令のフォーマット1（暫定版）

Iフィールド	未使用	未使用	未使用
1 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

ジャンプ先アドレス

命令の例

```
8000 0100     JMP 100H
```


プログラム開発用語集
日本語処理　全角文字は2バイトで表現されるため，日本語文字列操作では特別な関数に操作や判断が必要な場合がある．

ラムはなんとか書けそうです．ステップ4の処理がまだ残ってますが，これは非常にトリッキーなことをしてごまかします．

図10のようなプログラムでなんとか実現できるわけです．プログラムを停止させるために，HLT命令というものを定義する前に使っていますが，この命令でCPUは処理を止めるものと考えておいてください．トリッキーなところは，LOOPというラベルの付けられたLD命令の2語目を，プログラムの中で書き換えているところです．この2語目を2だけインクリメントすることで「次のデータ」のアドレスを得ることができるのです．2を加えるのは，ワード単位の処理だからです．PP-16では1語は2Byteになりますから．

このようなテクニックはいろいろと欠点があります．プログラムをROM化できないとか（ROMに入れたらプログラムは書き換えられませんから），マルチプロセスで動かす場合にコードの共有ができないとか，さらには書きにくい，修正ミスを起こしやすいなどの欠点です．

どうしたらよいでしょうか．

レベル3のまとめ

最低限のループが表現できるように，フラグを導入し，条件分岐命令を定義しました．

しかし，処理を行うメモリアドレスを次々に変化させていくプログラムを書くには，機能が不足しています．

課題

あなたの使っているCPUのアセンブラ命令について，分岐を行う命令（プログラム・カウンタの内容を変更する命令）を抜き出して，リストを作ってみましょう．

あなたが使っているCPUには，何種類のフラグがあるか調べてみましょう．

表2

ゼロフラグ	Z	結果が0
キャリーフラグ	C	加算の桁上がり
		減算の桁借り

図9　JMP命令のフォーマット2（条件分岐を可能にしたもの）

Iフィールド	CCフィールド	未使用	未使用
1 0 0 0		0 0 0 0	0 0 0 0

ジャンプ先アドレス

CCフィールドの値	略号	フラグとの対応	意味
0	F	0	常に不成立
1	T	1	常に成立
2	NZ/NE	!Z	等しくない !=
3	Z/EQ	Z	等しい =
4	CC	!C	>= （無符号）
5	CS	C	< （無符号）

図10

```
START:  LD      R0, C_ZERO      ; 総和をクリアしておく
        LD      R3, C_ONE       ; 後で使う定数をロードしておく
        LD      R4, C_TWO
        LD      R2, 10          ; R2はループカウンタ
        LD      R1, X_ADDR      ; ロードアドレスを初期化する
        ST      R1, LOOP + 1
LOOP:   LD      R1, X           ; X番地から始まるデータの一つをロードする
        ADD     R0, R1
        LD      R1, LOOP + 1    ; ロードすべきアドレスを+2する
        ADD     R1, R4
        ST      R1, LOOP + 1
        SUB     R2, R3          ; ループカウンタを -1
        JMP     NZ, LOOP
        HLT
;
C_ZERO: DW      0
C_ONE:  DW      1
C_TWO:  DW      2
X_ADDR: DW      X
;
X:      DS      10 * 2

        END
```

レベル4 アドレッシング

まだ，PP-16には，メモリ上のデータを読み書きする場合は，オペランドにアドレスの数値そのものを使う必要がありました．つまり絶対アドレッシングです．しかし，すでにレベル4で見たように，これでは非常に書きにくい処理があります．そこで，データのアクセス方法について，改良してみることにします．

処理すべきアドレスを指定するには，直接そのアドレスを指定するというのは，単純でハードウェアで処理をするにも楽ですし理解しやすいのですが，アドレスを可変するには不便です．C言語なんかでは，ポインタが使えましたよね．まずポインタを実装してみましょう．

ポインタというのは，データのアドレスが入っている変数と考えれば，だいたい雰囲気はつかめますよね．そこで，あるレジスタの内容を，データが含まれているメモリのアドレスと解釈して，そこからデータを読み書きすることにすれば，実現できそうです．

アセンブラ表記では，レジスタの中身をアドレスとして解釈する，という雰囲気をだすために()でくくることにして，

```
LD    R0, (R1)
```

のように表すことにしましょう．これで，R1の内容で示されるアドレスから1語読み出し，R0にロードすることを表します．命令形式は，未使用のフィールド3をAフィールドとして，絶対アドレッシングとポインタ・アドレッシングとの区別を行うようにします．また，フィールド4は補助のフィールド，Asフィールドとして，ポインタ・アドレッシングの時はポインタに使用するレジスタを示すことにします（図11）．ここではAはアドレッシング指定のA，AsはサブAフィールドのつもりです．

こうすれば，レベル3の目標となっているプログラムは，図12のように書き直すことができます．かなりすっきりしました．ほかに改良すべきところはあるでしょうか．

0や1といった定数をメモリ上に確保しておいて，そこからロードするというのは冗長ですね．定数をオペランドに書いて直接ロードできるなら，そのほうがすっきりします．このアドレッシング方法をイミディエート・アドレッシング（日本語なら即値アドレッシング）と呼びます．LD命令だけを拡張してもよいのですが（イミディエート・アドレッシングはST命令には無意味です），例のプログラムのようにロードした値はそのまま演算対象として使われることも多いので，2項演算命令を拡張したほうが筋がよさそうです．

その前に，2項演算命令の種類を増やしておきます．

今までは，加減算だけ定義していましたが，Cフラグを導入したので，多桁の計算ができるような加減算命令を定義します．ADCとSBBです．

図13のように，ソースレジスタを加減算するさいに，Cフラグも加減算を行うのが特徴です．筆算で多桁の加減算を行う場合は，下位の桁から順番に足していきますが，そのさい，より下位の桁からの桁上がりも足しこみます．同じ手法で，ADC命令を組み合わせれば，多倍長の加算ができます．

```
ADD    R0, R2
ADC    R1, R3
```

とすれば，R0とR1に分けて入れられた2語のデータに，同じくR2とR3に

図11　ポインタアドレッシンクを可能にするためのフィールド割り当て

MSB（15）～LSB（0）

15–12	11–8	7–4	3–0	
Iフィールド	Rフィールド	Aフィールド	Asフィールド	
0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1	LD R0, (R1)

Aフィールドの値	Asフィールドの意味	2語目の意味	説明
0	なし(0)	対象アドレス	絶対アドレッシング（対象アドレス）
1	ポインタ用レジスタ	なし	ポインタアドレッシング（レジスタ）

プログラム開発用語集
ネスト　複文・反復制御・選択制御構造・条件組み込みなどに存在する．ソースプログラムでは目で判断できるようインデントを行う．

分けて入れられていた2語のデータを足しこむことが可能になります．

さらに，比較をするためのCMP命令も付けましょう．演算はSUB命令と同じ減算ですが，結果をディスティネーション・レジスタに入れません．フラグだけが変化します．引き算をして，結果が0なら値は等しかったことがわかります．しかし，ただ調べるだけのつもりでSUB命令を使うと，レジスタの内容まで変化してしまいますから，不便です．そこで，こういう命令を新設したわけです．

あと，論理演算も追加しましょう．AND，OR，XOR，TST命令です．前の3種類はいうまでもないでしょう．TST命令は，結果をディスティネーション・レジスタに格納しないAND命令です．特定のビットが1になっているかどうかを調べるときにCMP命令と同じように使えます．

もう1種類，演算命令には見えませんが，レジスターレジスタ間のデータ転送命令のMOV命令も，2項演算の仲間として作っておきます．ソースレジスタに，「なにもしない」演算を施してディスティネーション・レジスタに入れるという解釈です．もちろん，ほかの一般のCPUでは，MOV命令はデータ転送命令であり，演算命令とは異なった命令フォーマットになることが多いと思います．ただ，PP-16の命令体系では，演算命令の仲間として扱います．以上の追加によって，演算の種類を表すIsフィールドの値は，図14のように割り当てることにします．

論理演算とMOV命令については，演算結果に桁上がりということは起きないので，Cフラグは常にクリアされることにします．ただし，Zフラグは影響を受けます．MOV命令でフラグが変化するというのも，演算命令として実装しているためにそのほうが統一がとれてしっくりくると考えているからです．が，この仕様が便利かどうかはちょっとわかりません．

これで，2項演算命令を充実させましたが，これにイミディエートで演算する値を指定できるように拡張します．イミディエートというより定数演算命

図12

```
START:  LD      R0, C_ZERO      ; 総和をクリアしておく
        LD      R3, C_ONE       ; 後で使う定数をロードしておく
        LD      R4, C_TWO
        LD      R2, 10          ; R2はループカウンタ
        LD      R5, X_ADDR      ; ポインタを初期化する
LOOP:   LD      R1, (R5)        ; X番地から始まるデータの一つをロードする
        ADD     R0, R1          ; 和をとる
        ADD     R5, R4          ; 次のアドレスを求める
        SUB     R2, R3          ; ループカウンタを -1
        JMP     NZ, LOOP
        HLT
;
C_ZERO: DW      0
C_ONE:  DW      1
C_TWO:  DW      2
X_ADDR: DW      X
;
X:      DS      10 * 2

        END
```

図13

```
SBB ADC     Rd, Rs        Rd = Rd + Rs + C
            Rd, Rs        Rd = Rd - Rs - C
```

図14　2項演算命令の拡張とフラグ変化の定義

Iフィールド	Rdフィールド	Isフィールド	Rsフィールド
0 1 0 0			

Isフィールドの値	命令	Z	C	N	V	演算内容
0	ADD	*	*	*	*	Rd = Rd + Rs
1	ADC	*	*	*	*	Rd = Rd + Rs + C
2	SUB	*	*	*	*	Rd = Rd - Rs
3	SBB	*	*	*	*	Rd = Rd - Rs - C
4	CMP	*	*	*	*	Rd - Rs （フラグだけ変化する）
5						
6						
7						
8	AND	*	0	*	0	Rd = Rd & Rs
9	OR	*	0	*	0	Rd = Rd ¦ Rs
A	XOR	*	0	*	0	Rd = Rd ^ Rs
B	TST	*	0	*	0	Rd & Rs （フラグだけ変化する）
C	MOV	*	0	*	0	Rd = Rs
D						
E						
F						

＊は変化，0はクリアされることを表す

命と呼んだほうがしっくりきますね。すでに2項演算命令の4つのフィールドは、すべて使っていますから、そのままの拡張は困難です。そこで、新たにIフィールドの値を割り当てて、新種の命令として扱います。Iフィールドの値は16進数のDを割り当てることにします。定数演算を行うための命令形式を図15に示します。この命令は2語から成り立ち、命令形式はほとんど2項演算のものと同じです。フィールドは、Iフィールド、Rdフィールド、Isフィールドが2項演算と同じように使われています（Iフィールドの値は違いますが）。ただし、Rsフィールドは使用せず、Rsの代わりに定数を直接2語目に配置します。アセンブラ表記では、その値自身を演算に利用することを明確にするために、値の前に#記号を付けることにしましょう。つまり、

```
ADD    R0, #10
```

などと表すわけです。こういう表現を使えば、問題のプログラムは、図16のようになります。これなら、そんなに無駄がないように見えます。とりあえずはこれで満足しましょう。

レベル4のまとめ

ポインタ・アドレッシングによって、メモリアクセスを容易にしました。

演算命令の種類を増やすと同時に、定数を利用する演算命令を新設しました。

課題

あなたの使っているCPUには、いったい何種類のアドレッシング・モードを用意してあるか、調べてみましょう。できれば、それら多くのアドレッシング・モードがどのような場合に役立つのか、考えてみましょう。

図15　定数演算命令のフォーマット（Is フィールドは2項演算命令と共通）

Iフィールド	Rdフィールド	Isフィールド	未使用
1 1 0 1			0 0 0 0
定数			

図16

```
START:  MOV     R0, #0          ; 総和をクリアしておく
        MOV     R2, #10         ; R2はループカウンタ
        MOV     R3, #X          ; ポインタを初期化する
LOOP:   LD      R1, (R3)        ; X番地から始まるデータの一つをロードする
        ADD     R0, R1          ; 和をとる
        ADD     R3, #2          ; 次のアドレスを求める
        SUB     R2, #1          ; ループカウンタを -1
        JMP     NZ, LOOP
        HLT
;
X:      DS      10 * 2
;
        END
```

レベル5
怒濤の拡張

この段階では、PP-16をさらに実用的にするために命令を拡張したり新設したりします。

まず、PP-16が扱うデータは、今まではすべて無符号の正の数であるとしました。しかし、負の数も必要となる場合もあるでしょう。符号付きのデータを表現するために2の補数表現を使うことにします。2の補数に関してはコラムを参照してください。2の補数では、最上位ビットが符号を表します。0なら正の値、1なら負の値です。そこで、演算後の結果の最上位ビットの値を反映する、Nフラグを新しく作ります。Nはネガティブの Nです。符号の意味であるサインの頭文字を取って、Sフラグと呼ぶCPUもありますが、

プログラム開発用語集
ヒストリ　履歴ともいう。キー入力や画面の変化などを過去にさかのぼって再現する機能。

さらに，場合によっては正の値と正の値を加えると最上位ビットまで桁上がりの影響が及んで，負の数と同じ表現になってしまう場合もあります．こんな場合にセットされるオーバーフローフラグ，Vフラグも新設します．NフラグとVフラグを正しく組み合わせると，符号付きの数の比較が可能になります．そこで，分岐命令のCCフィールドの取る値を，図17のように拡張します．

次に，バイト単位のデータを取り扱えるように命令を追加，拡張します．普通はアルファベットのような文字は1Byteのデータで表現されます．ASCIIコードなんかがよい例です．そこで，バイト単位でのデータ転送くらいはできないと，困ることが多いと思います．で，LD命令とST命令に対応する，バイト単位でのデータ転送命令を，LDB命令とSTB命令として新設することにします．それぞれ，Iフィールドの値は1と3に割り当てます．

図17　type4, 5命令のCCフィールド

CCフィールドの値	略号	フラグとの対応	意味
0	F	0	常に不成立
1	T	1	常に成立
2	NZ / NE	!Z	等しくない !=
3	Z / EQ	Z	等しい =
4	CC	!C	>= (無符号)
5	CS	C	< (無符号)
6	PL	!N	プラス
7	MI	N	マイナス
8	VC	!V	オーバーフローなし
9	VS	V	オーバーフロー
A	HI	! (C + Z)	> (無符号)
B	LS	C + Z	<= (無符号)
C	GE	! (N ^ V)	>= (符号付き)
D	LT	N ^ V	< (符号付き)
E	GT	! ((N ^ V) ¦ Z)	> (符号付き)
F	LE	(N ^ V) ¦ Z	<= (符号付き)

あとはLDやST命令とまったく同じです．あと，16bitレジスタの上位バイトと下位バイトを入れ替えることを可能にすれば，一通りのことは可能にな

COLUMN

PP-16のアセンブリ言語

本文では，PP-16のプログラムを示すために，すでにアセンブラを使うときのような表現をしています．多くのアセンブラで普通に使われている形式を利用しているつもりですが，念のため説明します．

PP-16のアセンブリ言語は行単位で意味を持ちます．1行は，最大4つのフィールドに別れます．ラベル，命令の略符号(ニーモニック)，オペランド，コメントです．

ラベルは，行の先頭から書き始め，コロン':'で終わります．ラベルは，そのラベルが配置された行がアセンブルされた結果，どの番地に配置されたかを知るための定数として振る舞います．

ニーモニックは，ほとんどはCPUの命令を表すための名前です．ただし，今回は，3種類だけ特別なアセンブラに対する命令，擬似命令（近ごろはディレクティブとか呼ぶのかな）があります．
オペランドは，命令の操作対象を指定するためのフィールドで，命令によって特定の個数だけ書き入れる必要があります．

コメント・フィールドは，セミコロン';'で始まり行の終わりまで続くフィールドで，その名の通りプログラムのコメントを入れるのに使われます．

CPUの命令については，プログラム例を示すときに説明しているはずですから，3種類の擬似命令だけ，ここで覚えておいてください．

END命令．プログラムの終わりを表します．これは文字どおりですね．

DW命令は，define wordの略です．1語単位の定数をメモリ上に配置するために使います．

DS命令は，define storageの略です．オペランドに指定された数値のバイト数だけ，メモリ・ブロックを確保するための命令です．後のほうのプログラムでちょっと出てきます．

るでしょう．MOV命令の変種として，データ転送をするさいに上下バイトを入れ替えるSWP命令（スワップ）を，2項演算の仲間に加えておきます．Isフィールドには16進数のDを割り当てます．

次にサブルーチン呼び出しができるように命令を新設します．サブルーチン呼び出しは，JMP命令に似ていてプログラムの別の場所へ飛んでしまいますが，しかるべき命令に出会うと呼び出した場所に戻ってくることができるような処理です．サブルーチン呼び出しの命令をJSR命令，戻ってくるための命令をRET命令と呼びましょう．サブルーチン呼び出しを行うには，戻ってくるアドレスをどこかに保存しておく必要があります．昔はいろいろな技法が使われていましたが，今ではスタックというデータ構造を使うことが普通になっています．

スタックは，先入れ後出しを基本とするデータ構造で，スタックポインタによって管理されています．スタックにデータを入れる時には，データをスタックポインタが指すメモリに入れてから，スタックポインタをデクリメントします．データをスタックから取り出すときには，スタックポインタをインクリメントしてからデータをスタックポインタの指すメモリから取り出します．スタックにデータを入れることをプッシュ，出すことをポップすると表現します（図18）．

具体的にサブルーチン呼び出しの手順を考えてみます．JSR命令では，まずJSR命令の次の命令を指しているプログラム・カウンタを，スタックにプッシュします．それから，ジャンプすべきアドレスを求めて，それをプログラム・カウンタにロードします．そうすれば次のタイミングで，新しいプログラム・カウンタの内容で示される場所から，命令を読みだして実行することができるわけです．反対にRET命令では，戻るべきアドレスを，スタックからポップすることで得て，それをプログラム・カウンタにロードします．

命令形式については，JSR命令はJMP命令と同じ形式にしましょう．ただしIフィールドの値は9を割り当てます．

また，RET命令はオペランドがありません．このような命令はまだ割り当てる場所がないので，新しいグループを作りましょう．特殊命令という名前でIフィールド7を割り当てることにします．今まで例題には使っていたHLT命令も同じ仲間です．HLT命令は，CPUを停止させる命令です．HLT命令が実行されると，CPUがリセットされるまで（あるいは，割り込みを受け付けるまで）まったくなにもしなくなります（注：このため，一般にOSの管理下でプログラムを実行する場合は，安易にHLT命令を実行するわけにはいきません．OSの動作まで停止してしまうシステムもあります．普通は，プログラムの最後にはOSに戻るための命令を置くことになります）．

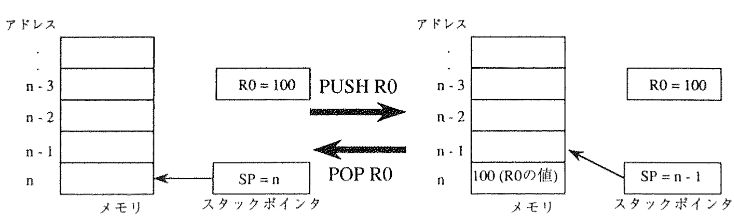
HLTは16進数で7000，RETは同じく7001というコードを割り当てます．

また，スタックはレジスタの内容を一時待避させる場合にも便利です．そこでレジスタの内容をスタックにプッシュしたり，スタックからポップするための命令，PUSH命令とPOP命令を作ります．アセンブラ表記なら，

```
PUSH    R0
POP     R0
```

なんていう形になります．MOV命令を2項演算とみなしたように，これらを単項演算とみなし，本来の単項演算であるビット毎の否定であるNOT命令や，単項マイナス演算子の役割を

図18　PUSH/POP命令の動作



PUSH命令を続けて使えば，n-1，n-2というアドレスに次々にデータを格納できる．POP命令で取り出す時は，最後にPUSHされたものから取り出す．

プログラム開発用語集
標準入出力　標準入出力ストリームの結合先．デフォルトではディスプレイとキーボードが指定されている．

するCPL命令を新設して，仲間に入れましょう．さらにシフト命令である，SHR命令とSHL命令も新設します．それぞれ，右シフトと左シフトです．レジスタからあふれたビットはCフラグに入れられます．

単項演算命令のＩフィールドは5を割り当てます．これら単項演算命令では，2項演算命令のときに利用した第4のフィールドであるRsフィールドを使用しません．そこで，シフト命令では，何ビットシフトするかをこの第4のフィールドで表現することにします．このフィールドをVフィールドと名付けることにします．

CPUの性能を上げるために，命令をできるだけ短くなるように設計することは，大切なことです．メモリ容量の制限がある場合には，命令が短ければ，それだけプログラムが小さくなりますし，命令をメモリから読みだす回数が減るぶんだけ高速に実行可能になります．レベル4での例題プログラムの最終形をみてもわかるとおり，定数の加減算は±1とか2程度の小さな数を使う場合がたいへんに多くなります．ポインタやカウンタの更新のほとんどはその程度の小さな単位について行われることでしょう．

そこで，単項演算子と同じＩフィールドに，短定数演算命令として，加減算を加えておきます．Vフィールドを，加減算する定数と解釈する加減算です．ですから，±15までの数値の加減算が1語で表現できます．いままでの定数演算命令形式では，2語必要でした．これで，Ｉフィールドが5の命令形式とIsフィールド，Vフィールドの意味は図19のようになります．

ところで，大切なことを忘れていました．スタックポインタという特殊なレジスタのことです．このレジスタは，少なくとも値を設定することが必要ですし，高級言語のパラメータ渡しなどに使うさいには，ほかのレジスタとの間で内容のやり取りが必要になります．そのために，スタックポインタを新設し，それを対象とした新しい命令を作ってもよいのですが，ここでは汎用レジスタのR15をスタックポインタとして使うことにしましょう．こうすれば，命令を新設する必要がなくなります．また，PP-16では汎用レジスタの数は16個と比較的多く，1つぐらい特殊な機能に使っても，不便にはなりません．

ほかの改良として，アドレッシング・モードを増やしましょう．すでにレベル4の課題で，アドレッシング・モードについて調べた方にはおわかりだと思いますが，現実のCPUには，多くのアドレッシング・モードが用意されていて，データに対する効率的なアクセス方法を（過剰装備なほど）充実させています．さすがにPP-16では，多種類のアドレッシング・モードを定義するつもりはありませんが，インデックス・アドレッシングとプログラムカウンタ相対アドレッシングぐらいは付けておいたほうがよいと思います．

インデックス・アドレッシングは，ポインタ・アドレッシングの変形で，レジスタの内容にオフセットと呼ばれる定数を加えたアドレスに対して操作を行うものです．オフセットはPP-16では16bitの1語を割り当てて，合計2語で命令を構成するようにします．構造体の内部や配列，あるいはコラム

図19　単項演算命令の定義

MSB 15 … 12	11 … 8	7 … 4	3 … 0 LSB
Ｉフィールド 0 1 0 1	Rdフィールド	Isフィールド	Vフィールド

Isフィールドの値	命令	Vフィールドの意味	Z	C	N	V	演算内容
0	NOT	なし (0)	*	0	*	0	Rd = ! Rd
1	CPL	なし (0)	*	0	*	0	Rd = - Rd
2							
3							
4	SHR	シフト回数	*	*	*	*	Rd = Rd >> n
5	SHL	シフト回数	*	*	*	*	Rd = Rd << n
6							
7							
8	ADQ	値	*	*	*	*	Rd = Rd + n
9	SUQ	値	*	*	*	*	Rd = Rd - n
A							
B							
C	PUSH	なし (0)	-	-	-	-	スタックへプッシュ
D	POP	なし (0)	-	-	-	-	スタックからポップ
E							
F							

プログラム開発用語集
標準補助入出力　標準補助入出力ストリームの結合先．デフォルトではRS-232Cポートが指定されている．

で触れるようにスタック上に確保したデータにアクセスする場合に便利です.

また，プログラムカウンタ相対アドレッシングは，プログラムカウンタを対象にしたインデックス・アドレッシングです．プログラムカウンタの値にオフセットを加えたアドレスのデータを操作対象にするものです.

JMP命令とJSR命令のフィールド3とフィールド4がまだ未使用でしたから，これをLD命令などと同じAフィールドとAsフィールドに利用することにします．そうすれば，ジャンプするアドレスを指定するのに，便利になります.

レベル5のまとめ

符号付きの数値を扱えるように，フラグを追加し，大小比較を可能にしました.

サブルーチン呼び出し，スタック操作が可能な命令を新設しました.

アドレッシング・モードを強化し，メモリアドレスをオペランドにもつ命令すべてに使えるようにしました.

バイト単位のデータ転送命令を定義して，文字処理など，一応可能なようにしました.

そのほか，演算命令などを増やしました.

この時点での，最終的な命令一覧を表3に，命令形式のまとめを図20に示します.

レベル6 反省と拡張の可能性

とりあえず，今回の命令の検討はこれで終りにします．一応，これだけあれば，それなりのプログラムが作成可能なはずです．しかし，まだこのPP-16の命令体系には，ずいぶんな無駄と，不足する点があります．お気づきでしょうか.

不足する点から調べましょうか．ま

COLUMN

スタックとローカル変数

サブルーチンの中で用いるローカル変数をメモリのどこに配置するか，という問題にもいくつかの解決方法があります．CやPascalなどの高級言語のコンパイラが採用している方法は，スタックの中に確保するというものです．再帰的なサブルーチン呼び出しが可能になるとか，サブルーチンからリターンするときにはデータ領域が開放されるのでデータ記憶用のメモリが少なくてすむとか，プログラム領域とデータ領域が自然に分離されるなどの特徴があり，優れている点が多いと考えられます．スタックの利用法の例として，確認しておきましょう.

ここにxとyという単純なローカル変数を2つ必要とするサブルーチンがあるとします．この変数をスタック上に配置するには，次のようなコードをサブルーチンの先頭に置けばよいのです.

```
PUSH    R14
MOV     R14, R15
SUB     R15, #2
```

これで，R14の内容を保存した上で，スタック上に2語の空きスペースを作り出すことができました．この後は，R14の指すメモリをxと，R14の内容-1の番地のメモリをyとみなせばよいのです．xからyに値をコピーしたいときには，たとえばR0を一時的に用いて

```
LD  R0, (R14)
LD  -1(R14), R0
;インデックス・アドレッシング
```

とすればできます．配列の一要素にアクセスするときばかりでなく，このような場合にもインデックス・アドレッシングは便利です.

なお，このサブルーチンからリターンするときには，必ずスタックを元に戻す必要があります．つまり，

```
MOV     R15, R14
POP     R14
RET
```

というプログラムになります.

サブルーチン呼び出しのときの引数をスタックにプッシュして受け渡しをすることも行われますが，その場合も同じ手段で，R14を基準にしたインデックス・アドレッシングでアクセスすることができます.

なお，なぜ直接スタックポインタであるR15を基準にしたインデックス・アドレッシングを使わずに，R14にR15の値をコピーしてから使うかといえば，サブルーチンの中で一時的にPUSH命令などのスタック操作を行うとR15の内容が変化してしまうからです，念のため.

プログラム開発用語集
ファンクションコール　MS-DOSシステムコールのひとつで，INT 21Hをさす．AHレジスタに機能（ファンクション）番号を入れ呼び出すことで多くの機能を実現する.

図20　PP-16命令形式のまとめ

Iフィールドの値と命令の対応

Iフィールドの値	命令内容	命令形式
0	LD	type 1
1	LDB	type 1
2	ST	type 1
3	STB	type 1
4	2項演算	type 2
5	単項演算 短定数命令	type 3
6		
7	特殊	type 6
8	JMP	type 4
9	JSR	type 4
A		
B		
C		
D	定数演算	type 5
E		
F		

type 2, 5命令のIsフィールドと演算

Isフィールドの値	命令	Z	C	N	V	演算内容
0	ADD	*	*	*	*	Rd = Rd + Rs
1	ADC	*	*	*	*	Rd = Rd + Rs + C
2	SUB	*	*	*	*	Rd = Rd - Rs
3	SBB	*	*	*	*	Rd = Rd - Rs - C
4	CMP	*	*	*	*	Rd - Rs （フラグだけ変化する）
5						
6						
7						
8	AND	*	0	*	0	Rd = Rd & Rs
9	OR	*	0	*	0	Rd = Rd ¦ Rs
A	XOR	*	0	*	0	Rd = Rd ^ Rs
B	TST	*	0	*	0	Rd & Rs （フラグだけ変化する）
C	MOV	*	0	*	0	Rd = Rs
D	SWP	*	0	*	0	Rd = (Rsの上下バイトを入れ換えたもの)
E						
F						

*　結果によってフラグが変化する
0　フラグがクリアされる
　変化しない

type 4, 5命令のCCフィールド

CCフィールドの値	略号	フラグとの対応	意味
0	F	0	常に不成立
1	T	1	常に成立
2	NZ / NE	!Z	等しくない !=
3	Z / EQ	Z	等しい =
4	CC	!C	>= （無符号）
5	CS	C	< （無符号）
6	PL	!N	プラス
7	MI	N	マイナス
8	VC	!V	オーバーフローなし
9	VS	V	オーバーフロー
A	HI	! (C + Z)	> （無符号）
B	LS	C + Z	<= （無符号）
C	GE	! (N ^ V)	>= （符号付き）
D	LT	N ^ V	< （符号付き）
E	GT	! ((N ^ V) ¦ Z)	> （符号付き）
F	LE	(N ^ V) ¦ Z	<= （符号付き）

type 3命令のIsフィールドとVフィールド

Isフィールドの値	命令	Vフィールドの意味	Z	C	N	V	演算内容
0	NOT	なし (0)	*	0	*	0	Rd = ! Rd
1	CPL	なし (0)	*	0	*	0	Rd = - Rd
2							
3							
4	SHR	シフト回数	*	*	*	*	Rd = Rd >> n
5	SHL	シフト回数	*	*	*	*	Rd = Rd << n
6							
7							
8	ADQ	値	*	*	*	*	Rd = Rd + n
9	SUQ	値	*	*	*	*	Rd = Rd - n
A							
B							
C	PUSH	なし (0)	-	-	-	-	スタックへプッシュ
D	POP	なし (0)	-	-	-	-	スタックからポップ
E							
F							

type 1, 4命令のAフィールドとAsフィールド

Aフィールドの値	Asフィールドの意味	2語目の意味	説明
0	なし (0)	対象アドレス	絶対アドレッシング（対象アドレス）
1	ポインタ用レジスタ	なし	ポインタアドレッシング（レジスタ）
2	インデックスレジスタ	オフセット値	インデックス・アドレッシング（レジスタ + オフセット）
3	なし (0)	オフセット値	相対アドレッシング（プログラムカウンタ + オフセット）
4			
5			
6			
7			
8			
9			
A			
B			
C			
D			
E			
F			

type 6 命令の命令割り当て

命令コード	命令	意味
7000	HLT	CPUの動作を停止する
7001	RET	サブルーチンから戻る PC = Mem[++R15]

プログラム開発用語集
プリプロセッサ　ソースファイルの一部分をある条件で加工したり，読み飛ばしたり，ほかのソースファイルを読み込むプログラム．

表3　PP-16の全命令表

ニーモニック	type/I フィールド	動作
ADC	2/4(5/D)	Rd = Rd ＋ Rs(const)＋C
ADD	2/4(5/D)	Rd = Rd ＋ Rs(const)
ADQ	3/5	Rd = Rd ＋ shortConst
AND	2/4(5/D)	Rd = Rd & Rs(const)
CMP	2/4(5/D)	Rd - Rs(const)
CPL	3/5	Rd = - Rd
HLT	6/7	CPU Halt
JMP	4/8	Jump
JSR	4/9	Jump to Subroutine
LD	1/0	Load to registor
LDB	1/1	Load one Byte to registor
MOV	2/4(5/D)	Rd = Rs(const)
NOT	3/5	Rd = ! Rd
OR	2/4(5/D)	Rd = Rd | Rs(const)
POP	3/5	Rd = Mem[++R15]
PUSH	3/5	Mem[R15--] = Rd
RET	6/7	Return from Subroutine(PC = Mem[++R15])
SBB	2/4(5/D)	Rd = Rd - Rs(const) - C
SHL	3/5	Rd = Rd << shortConst
SHR	3/5	Rd = Rd >> shortConst
ST	1/2	Store from registor
STB	1/3	Store one Byte from registor
SUB	2/4(5/D)	Rd = Rd - Rs(const)
SUQ	3/5	Rd = Rd - shortConst
SWP	2/4(5/D)	Rd.hi = Rs.low, Rd.low = Rs.hi
TST	2/4(5/D)	Rd & Rs(const)
XOR	2/4(5/D)	Rd = Rd　Rs(const)

注：type/I フィールドの項で () で囲んであるものは，オペランドにRsの代わりに定数を使う場合の分類．shortConstは0から15までの定数．

ず，バイト単位のデータ形式の処理能力が不足しています．

一応，1Byteのデータをレジスタにロードできても，演算関係はまだ1Byteだけの処理はできません．たとえば，1Byte単位での比較を行おうとしても，ロードしたときにレジスタの上位バイトに残っているゴミデータのために，等しいかどうかの判定が狂う可能性があります．姑息にバイト単位での演算命令を，Iフィールドに別の値を割り当ててつけ加えてもなんとかなります．しかし本来なら，最初から数種類のデータ長に対応できるようなデータの種別を指定するフィールドを加えていたほうがすっきりするでしょう．

あと，アドレッシング・モードが比較的単純で，2語長の命令頻度が多くなっている点は欠点です．できれば，命令の第1語の一部に，オフセット値などを埋め込んだ1語命令によるアドレッシング・モードがあれば命令が短くなって性能が上がると考えられます．

総合的には，レジスタ数を8個に減らして，レジスタ指定用のフィールドを3bitにしてしまえば，演算命令においてもデータ長を指定するフィールドを新設できますし，1語命令を増やすこともできそうで，性能が上がるかもしれません．このように，レジスタ数というのも，多ければ多いほどトータルの性能がよくなるというわけではないのです．レジスタ数が多ければ，それだけ命令の中でレジスタを指定するフィールド幅が必要になり，多ワードの命令が多くなってしまって，命令をフェッチするための時間がよけいに必要になって性能が落ちる可能性があります．もちろん，汎用レジスタが手ごろな数だけ合ったほうが，プログラミングを行うときには楽です．

こういった問題の折衷案として，特定のレジスタ（たとえばR0だけ）をディスティネーション・レジスタとする演算命令だけは，特別の短縮形式の命令(たとえば1Byte)を用意する方法があります．レジスタを1つしか使わない場合にはそれなりに短くて高速なプログラムを作成可能になり，汎用レジスタを同時に幾つも使いたい場合は通常の1ないし数語の命令を若干の時間的ペナルティを払ってプログラムするという使い分けができます（たとえば80x86系のCPU)．おっと，いつのまにか過剰仕様の話になってしまいましたね．

ほかに無駄で損をしているといえば，JSR命令にまでCCフィールドがあって，条件分岐できるのは過剰仕様のような気がします．サブルーチン呼び出しに条件指定ができても，実際にはほとんど使わないでしょう．その分，別の高度な命令にフィールドを割り当てたほうが得をするような気がします．とくに，条件分岐命令というのは，1つのサブルーチンなり関数なりのモジュール内だけで使われるものですから，1KByteも離れたところにジャンプするのは希です．それなら，条件分岐命令は8bitのオフセット，つまり＋127Byte-128Byteの範囲内にプログラムカウンタ相対アドレッシングで分岐するようにしたほうが1語におさまって効率が高いように思えます．JMP命令やJSR命令からCCフィールドを省略して，無条件分岐専用の命令に

プログラム開発用語集
ブレークポイント　デバッガのターゲットプログラム中で，あらかじめ動作を停止するポイント（アドレス・ラベル）のこと．

して，条件分岐には短縮形式のBR命令（ブランチ→分岐のつもり）だけを使うわけです．どうしても遠くまで条件分岐したい場合には，一度近所に遠くの目的地へ分岐するJMP命令を置いて，そのJMP命令へBR命令で分岐するようにします．

ほかにPP-16の命令体系で混乱しているところがあると考えられるのは，MOV命令のフラグの扱いです．ちょうど演算命令の1種として割り当てると美しいかと思ったのですが，そうするとフラグが変化してしまいます．

とにかく，CPUの命令体系の設計というものは，バランス感覚が大切です．演算の種類やアドレッシング・モードは多種類の高機能なものを用意してほしいという欲求がつねにあり，また一方で命令自身は短いほうが処理速度が上がるうえにプログラムサイズが小さくなるからありがたいというふたつの力の危ういバランスのうえに決められていくのです．そういう意味では，このPP-16の命令体系は失格です．単純さ命で，実行効率のことをほとんど考えていませんでした．

メーカーが命令体系を設計する際には，命令体系の候補をいくつも設計しながら，いくつかのアプリケーションを作成してみたり，コンパイラが効率のよいコードを出力できるか，実際にコンパイラのコード生成部を作成して実験したり，多くの検討を加えて絞りこむのです．もちろん莫大な費用と時間が必要になります．

そうして，十分に優れた命令体系が得られたとしてもまだ問題があります．とくに現実のCPUですと，初期の命令体系の設計がそのまま何年も使われていくことはありません．などというと誤解されそうですが，別に同じCPUを使っているのに命令体系が変わってくるということはなく，現実には，半導体の設計製造技術の進歩によって，さらに大規模で高速なCPUが設計可能になります．すでに市場で使われているCPUにとっては，その命令体系にしたがって作成されたプログラムというのは財産になります．より高機能なCPUが作成可能になっても，それ以前のプログラムはできるだけそのまま動いてほしいという希望はありますが，それと同時にさらに高機能な命令やアドレッシング・モードを追加してほしいという要求もあります．

これらをバランスよく満足するには，初めの段階から命令の拡張計画を考えて体系を設計するか，それまでに使われていなかった命令コードになんとか追加命令を押し込んでしまうかという判断をしなければなりません．場合によっては，命令体系を一新してしまい，どうしても過去のCPUの命令を実行したいという人のためのエミュレーション・モードを付けてお茶を濁すとか，いっそのことときっぱり過去のしがらみを切り放すかしたほうが価値の高いものが産まれる場合だって考えられます（モトローラ社などのRISCチップなどは最後の例と考えられますね）．

現在のほとんどの32bit CPUは，すべてこのような複雑な要求のバランスをそれぞれできるだけ満たすように設計され拡張されてきています．それだけ，複雑で混乱しているように見える命令体系になってしまうわけです．

ところで今回考えたCPUの命令は，転送命令，演算命令，分岐（条件付き，無条件を含む）命令に大きく分類できます．実際のCPUの命令は，ほかにOSサポートのための命令など，これ以外の範疇のものも種類だけは多いのですが，実際のプログラミングで多く使われている命令はこれら3種類の中の一部です．あと，アドレッシング・モードもまとめて，ノートなりを作成すれば，アセンブラレベルでの命令の理解はほとんど終わっています．命令を表にして分類する際に，命令のフィールド構成とかバイト数に着目して，設計者が「この命令は数多く使われるだろう」とか「効率を上げるために用意した短縮形の命令」と考えているらしいことがわかれば，それを表の上のほうに持ってきて優先的に見つけられるようにします．PP-16で見たように，演算命令は数は多くても，ほとんど命令フォーマットやフラグの変化などは共通の場合が多いので，それで覚えることがぐんと減ります．ただ小さな数の加減算は短縮形式が用意されたりしていますが，それだけ追加して理解すればよいわけです．設計者はそれなりに体系的に設計しているはずですから．

そんなわけで，すでに特定のCPUに対するアセンブラレベルでのプログラミングの経験があれば，今まで体験したことのないCPUのアセンブラにチャレンジするときでもこれだけを自分で整理すれば，かなりのことができるようになるはずです．とくに組み込み用途の8bitクラスのCPUを扱うときは，ほとんど逆アセンブラを作れるくらいまで資料を自分なりにまとめれば，自在に命令を扱えるレベルまで達していると思います．

最後に課題をあげますので，できれば試しに考えてみてください．

PP-16を32bit CPUに拡張したい．レジスタ構成や演算命令などを拡張する方法をなん通りか考えてみましょう．特にデータが32bit整数，16bit整数，8bit整数の3種類の長さの演算をサポートしなければなりません．また，シフト命令などのシフト回数の指定法な

プログラム開発用語集
プロシージャ　プログラム本体とは別に定義したサブルーチンを，とくにプロシージャという．C言語でいえば関数にあたる．

ども変更する必要があるでしょう．

PP-16のメモリ空間を1MByte (20bit) にする場合には，どのような拡張を施すのがよいでしょうか．あるいは4GByte (32bit) に拡張する場合はどうでしょう．アドレッシング・モードの拡張方法，さらにインデックスレジスタを32bitにするためにすべての汎用レジスタを32bit長にするのが良いかどうか，など現実のCPUと比較しながら考えてみましょう．

PP-16の命令体系が不十分だと思えてきたら，自分で最初から16bitクラスのCPUの命令体系を設計してみましょう．設計が終わったら，はたしてどれくらい使える命令体系か，実際にいくつかプログラムを書いてほかの市販CPUなどと比較してみること．たとえば，文字列の一致検査や整数のソートなどのプログラムがよいでしょう．

COLUMN

アセンブラと逆アセンブラ，エミュレータ

先月号でもアセンブラの作り方が特集されていましたので，詳しくはそちらを参照していただくことにして，ここでは簡単に触れておきます．

マクロ機能やリロケータブル・オブジェクトの出力を考えない最低限のアセンブラは，ニーモニックで書かれた命令コードを機械命令に変換することとラベルの処理がおもな仕事になります．ニーモニックを機械命令に変換する部分は，図20のようなテーブルを作成しておけば単なる文字列検索になりますし，オペランドの処理は命令のタイプ別に処理すれば終わります．機械命令のフォーマットがニーモニックだけで決まらずにオペランドの形式で変化する場合は，ちょっと手間がかかりますがそういう特別な命令タイプがあるようにコーディングすればなんとかなります．

386やMC68030などの最近の32bit CPUのアセンブラでは，アドレッシング・モードが多様で，その解析が非常に複雑な作業になりますが，8bit CPUではそんなことも少なく比較的簡単にアセンブラを作成することができるでしょう．

逆アセンブラも，市販品には，ラベル処理を行うなどアセンブラの入力にできるようなレベルの変換能力，データとコード部分の分離の自動化など，高機能なものが多いのですが，symdebに内蔵されている程度の逆アセンブラなら，やはり図20程度の表を作成できればすぐにプログラミングできます．

PP-16の逆アセンブラを作成することを例に考えます．

機械命令からニーモニックに変換する場合，ロード／ストア／ジャンプ系の命令ならIフィールドだけで命令を特定できます．演算命令なら，Iフィールドとlsフィールドで命令を特定できます．ですから，命令を特定するのに必要なビットパターンを表に加えておいて，そのデータと機械命令のANDをとってから命令の基本形と比較を行い，ニーモニックとオペランド変換のしかたを特定します．Cで基本的な部分を作成すると，作成しておくべき表のデータはリストAのような形になります．変数opcodeで示される命令が，上の表の1つのエントリに一致するか調べるコードは，リストBのようになります．一致したことがわかれば，typeフィールドに応じてレジスタやアドレッシング・モードの解析を行い，mnフィールドのニーモニックと合わせて出力をすれば逆アセンブラの完成です．

CPUによっては，PP-16のADQ R0, #0のように無意味な命令パターンを，別の特殊な命令に割り当ててあることも多いのですが，そのような場合はそちらの命令を先に検出するようにプログラムすればよいのです．optabを頭から線形探索する場合は，maskフィールドの1になっているビット数が多いものほど表の先頭に近いところにあるようにすれば，目的は達せられます．

エミュレータは，機械命令をインタープリトするプログラムです．プログラムをデバッグしたりするのに使われたりしますが，これも逆アセンブラとほぼ同じ作り方ができます．逆アセンブラは命令を解析してアセンブラ風の文字列を出力しますが，エミュレータは命令を解析して，文字列を出力する代わりに即座に対応する動作を行えばよいのです．命令の解析法は逆アセンブラと同じものが使えます．メモリやレジスタに対応するものは，配列や変数として定義します．

プログラミング・ツールの貧弱なはるか昔の頃は，逆アセンブラなどを作成する人も多かったのですが，最近はその必要性も薄れていますね．ただ，初見のCPUの命令体系を勉強するには，単にインストラクション・セットを眺めるより，optab[]相当の表を作成してみることで短くて効率の良いプログラムを書くためのコツを得ることができたりして，なかなかよいことだと思います．

リストA

```
struct {
        int mask;       /* 命令の骨組みを抜き出すマスク */
        int skeleton;   /* 命令の骨組み                 */
        int type;       /* オペランドの変換のしかた     */
        char *mn;       /* 命令のニーモニック           */
} optab[] = {
        { 0xf0f0, 0x4000, 0, "ADD"},
                  .
                  .
                  .
        { 0xf000, 0x2000, 1, "ST"},
                  .
                  .
};
```

リストB

```
if((optab[i].mask & opcode) == optab[i].skeleton) {
        /* 一致した      */
}
```


プログラム開発用語集
フローチャート　処理や作業の手順を決められた記号を使って表記した図で，処理の流れが視覚的に理解できる．

特集
過激なプログラミング入門

すでに簡単なCPUの命令セットを設計者の目から眺めてきた．なぜそういう命令を定義する必要があったかという意図については，ある程度はおわかりになったかと思う．ほぼ命令セットが固まったら，次はハードウェアの設計にかかる．そういうわけで，前回で決まった命令セットを持つPR[illegible]のハードウェアの構成法を考えてみよう．

プログラミングの理解は，まずハードウェア設計から

石からはじめるプログラミング その2

虎岩　登

まずデータの流れから

本題に入る前に指摘しておきますが，CPUの設計にとって，もっとも困難でかつ設計者としても力を注ぐ場所というのは，命令セットの定義です．必要十分な命令セットをきちんと設計するのは非常にむずかしいことです．また，ハードウェアでどのように実現するかを横目で見ながら，どのようなプログラムを動かすためのCPUか，そのCPUが実行するOSや応用プログラムの種類を考え，またどのようなコンパイラによってそれらのプログラムが作成されるかなど，本当はきちんと考えなければいけません．COBOLばかりが使われるなら10進数（BCD）の演算命令を多く取り入れれば性能が上がるでしょう．マルチタスクのOSなら，コンテキスト・スイッチに関する命令を高速にするべきかもしれません．また，コンパイラの最適化アルゴリズムなども調べて，コンパイラがコードを出力しやすいような命令体系にするのも，全体の性能を上げるには効果的かもしれません．

単にどのような命令体系が有利か，想像するだけでなく，実際にコンパイラや応用プログラムを作成して，実際にもっとも高性能な命令体系を「測定」する必要があります．

そのような苦労を考えれば，ハードウェア設計，特に最低レベルでのとにかく命令をハードウェアで実行できればいいというハードウェアなら，それなりに体系的に実現することができます．もちろん，これはハードウェア屋の楽しみというか苦しみというか，とにかく仕事なわけですが，ソフトウェアを書くにあたっても，単なるブラックボックスのハードウェアを操作するより，ある程度は中で何が行われているかわかっているほうが安心できるかもしれません．すくなくとも，足が地についた気がするでしょう．

TV受像機の動作に興味を覚えた人が，いきなり回路図を見ても意味がないと思います．しかし，放送局から送信されている電波の形式，信号フォーマットを調べて，TV受像機の中の回路モジュールがその信号フォーマットをどのように変形してブラウン管に表示するかを学ぶことは可能ですし，十分に意味があることだと思います．それと同じことで，今回はハードウェアの詳細については触れず，CPUの中のモジュール構成とそのモジュール間のデータの流れに着目して説明します．概念的な視点からだけですが，PP-16の実現を例にあげてハードウェアの中でどのようなことが行われているか見ていきます．

レベル1 前提とするハードウェアと表記法

最初にいきなり，PP-16で前提とするハードウェア構成を図1に示します．比較的小さなハードウェア規模で実現しようと，単純な構成に絞ってあります．入り口に近いところから順番に見ていきましょう．

データの入出力を行うところには，データバッファ回路DBがあって，CPUの外部と内部のデータバスを分離しています．Iレジスタは，CPUの外から読み込んだ命令を記憶するための特殊なレジスタです．このほかにも，いくつかプログラマからは操作できないハードウェア処理のための特殊レジスタが，CPUの中に存在します．ハードウェアで実装する場合も，一時的にデータを記憶することが必要な場合がままありますから，このようなレジスタがいるのです．

Iレジスタに記憶された命令は，命令解読モジュールに送られます．ここでは，命令を分析して，命令の種別ごとにどのような動作を行うべきかを示す信号を作り出します．

レジスタ・ファイルには，R0からR15までが並んでいます．本稿では，特定の汎用レジスタを表すのにR[n]という表現も許すことにします．つまりR[3]はR3と同じものです．

演算はALU（算術論理演算ユニット）で行われます．プログラムカウンタも内部データバスに接続されています．ALUの入力側には，バッファ用のレジスタとして，XレジスタとYレジスタがあります．

ほかにテンポラリレジスタとしてTレジスタがあります．内部データバスにはアドレス出力の値を保持するAレジスタもついています．Aレジスタの出力は外部アドレスバスに出されています．

ほかには，全体のタイミングを制御する制御回路があり，これら回路全体の制御を司っています．

この回路構成は，入門書に出てくるコンピュータの構成とどことなく似て

プログラム開発用語集
ページング　仮想記憶システムで，外部ページ記憶装置上のプログラムをページ単位に分割すること．

います．データバッファ回路とAレジスタがコンピュータの入出力装置，レジスタファイルがコンピュータのメモリ，ALUがコンピュータ全体でいうCPUなどのように対応が付けられそうです．実際，命令解読やタイミング制御などには「マイクロプログラム」というプログラム制御方式が使われることもありますから，一種の小さなコンピュータとして考えてもそれほど不自然ではありません．

では，このハードウェアで命令がどのように実行されるか見ていきます．すでにお話したようにデータの流れに着目して説明していきますから，その表記法を決めておきましょう．Aという場所からBという場所へデータ転送をする場合は，

A→ B

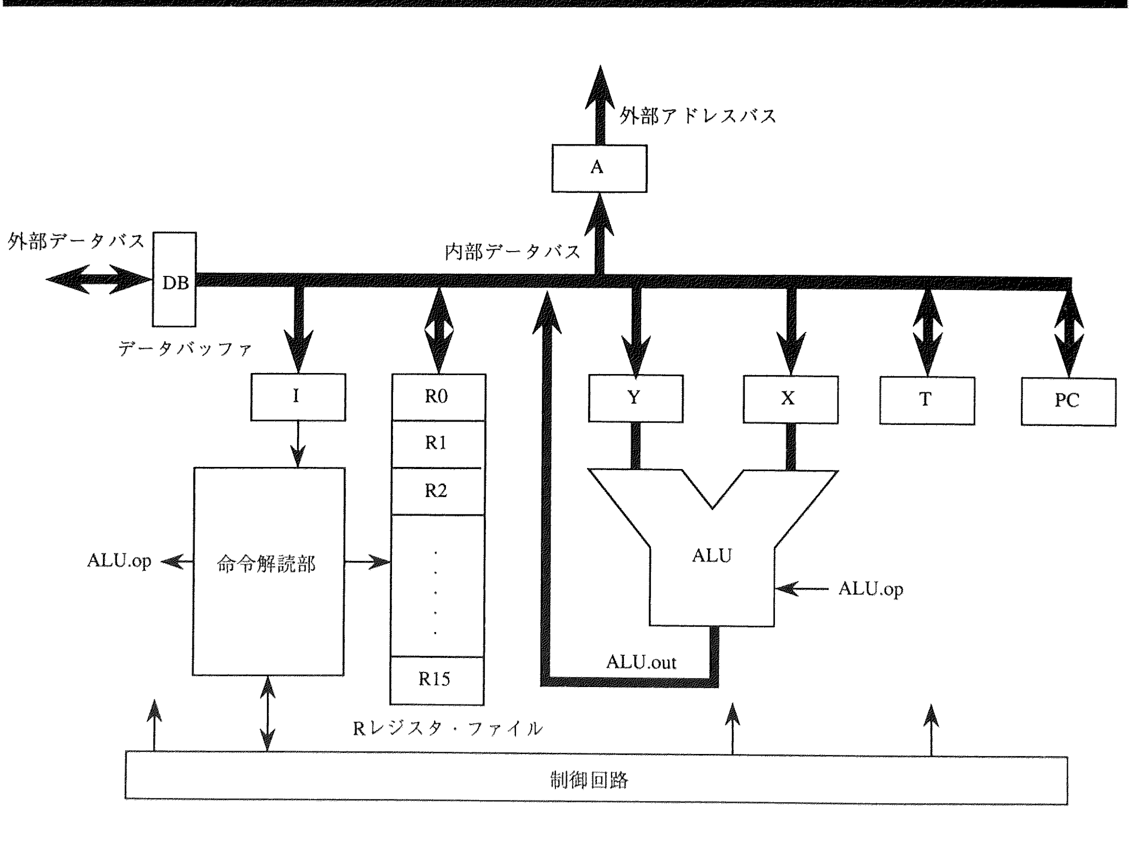
と，表現します．AからBへ転送したデータをさらにCに転送する場合は，

A→ B
B→ C

と表現します．ただし，1行は1クロックで実行されると考えてください．おっと，その前にクロックというタイミングの基準信号について説明しなければいけませんね．

デジタル回路では，ほとんどの場合「クロック信号」という一定の周期のパルス信号を使ってタイミングをとります．音楽にたとえましょう．いくつもの楽器が1つの曲を演奏するとき，それぞれの楽器の奏者が勝手なテンポで演奏したら，曲としては聞こえません．テンポを決めてやる何かが必要になります．デジタル回路でも，全体の機能を実現するには多くのモジュールを組み合わせる必要があります．それぞれのモジュールには単純なものも複雑な回路構成のものもいろいろありますから，データ処理をする時間も一定ではありません．各モジュールを勝手な速度で走らせてしまったら，あるモジュールが送り出していないデータに対して演算を施したり，逆にまだ後ろのほうがつかえているのに続けてデータを送り出したり，まともな処理ができません．そこで，クロック信号というすべてのモジュールが一定の処理を行うのに十分な時間間隔を保証された信号を用意して（すなわち一番遅いモジュールに合わせて），クロック信号が

図1　PP-16のブロック図(1)



変化すると同時に各モジュール間でデータの受け渡しをすることにします．

つまり,「このコンピュータのクロックは20MHzの386だ」とかいうときのクロックのことです．一般に，クロック信号が速くなれば，すなわちクロック周波数が高くなれば，コンピュータの速度もそれに比例して速くなります．ただし，これは同一の回路構成の場合だけで，そもそも回路構成が異なり1クロック単位で処理される内容が違っていれば単純に比較することはできません．

高速になるからといって，クロック周波数を勝手に高くしてもよいかというと，クロック信号というのは内部のモジュールの一番遅いものでも間に合うようにしてやらなければ回路の正常な動作ができませんから，そう簡単なことではありません．回路の動作速度というのは，製造上のばらつきや動作時の温度・電源電圧などで変化します．最悪の条件でも安定して動作させるために，エレベータのワイヤーロープのように十分なマージンを取る必要があります．

というわけで，今回は回路の動作を示すときには1行が1クロックで処理されるように表現します．ハードウェアでは，自然に並列処理ができます．別々のデータ転送経路を持っていれば，1クロックで同時にデータ転送を行うことも可能です．AからBへ転送するのとXからYへ転送するのを同時に行うときは

A→ B, X→ Y

と表現することにします．

データ転送以外のなんらかの処理は，普通に日本語表記することにしましょう．

レベル2 演算命令

では，最初に演算命令，特に2項演算を中心にCPUの中で何が行われているか考えます．

まず，命令をメモリから読み出さなければ，命令の実行はできません．このためには，図2のような手順を踏みます．これで，プログラムカウンタの示すメモリの内容がIレジスタに転送されます．この手順はすべての種類の命令の読み出しに共通です．図1には示しませんでしたが，プログラムカウンタには，その内容をインクリメントする専用回路がついているものとします．なにせ分岐命令を実行しなければ，常にプラス1を行いますから専用回路をつけておいても損ではないでしょう．

「メモリからデータが出力される」という表現は，これからは省略して「メモリアクセス」とします．また，この表記だと，メモリからデータが出力されるのは1クロックということになっていますが，実際には外部のメモリアクセスはCPUの内部処理速度より遅いので，数クロックかかるのが普通です．

さて，これでIレジスタに命令がセットされました．ここから命令解読モジュールが動作します．PP-16の命令体系だと，Iフィールドを調べることで，どのような種類の命令かわかります．Iフィールドが4であれば，2項演算命令です．2項演算命令とわかれば，どのレジスタに対しての計算か，RdフィールドとRsフィールドでわかりますし，ALUに与える演算の種別を表すコードはIsフィールドにあ

図2　メモリから命令を読み出す

```
PC → A
PC++, メモリからデータが出力される
DB → I
```

図3　2項演算命令

```
R[Rd] → X
R[Rs] → Y, Is → ALU.op
ALUで演算, ALU.out → R[Rd]
```

図4　2項演算命令のまとめ

```
PC → A
PC++, メモリアクセス
DB → I
R[Rd] → X
R[Rs] → Y, Is → ALU.op
ALUで演算, ALU.out → R[Rd]
```


プログラム開発用語集
マルチモジュール　複数のソースファイルから，ひとつの実行形式を作成すること．モジュール単位でアセンブルできるので大規模なプログラムでは必須

ります．そこで，内部では図3のような動作が行われます．

ALUの演算を指定する入力をALU.op，ALUの出力をALU.outと表記しました．データ転送経路が内部データバスだけですから，これ以上の並列化はむりです．ALU.opだけは別の転送経路が確保されています（図1）．

まとめると，図4のような形になります．

イミディエート・アドレッシングでの演算命令，ADD R1，#100のような定数演算命令はどうなるでしょうか．これもIフィールドで命令がすぐにわかります．ただしプログラムカウンタを使って数値を余分に読み出さなければなりませんから，図5のようになります．2番目のメモリアクセスの時，内部データバスは空いていたので，R[Rd] → Xというデータ転送を同時に行うことにしました．並列動作が容易であるというハードウェアの特徴を活かしているわけです．

単項演算なども同じ流儀で実現できます．

また，PP-16流のMOV命令の場合は，ALUの演算として，ALU.out = Rsとなるものを用意しておけば，すでに述べた内部処理でそのまま実現できます．ただし，どうしてもこの段階でチューンしておきたいというなら，R[Rd] → Xの操作を省略するように，制御回路に細工をするべきでしょう．コストと手間をとるか，性能をとるか，設計者の判断にゆだねられる1つの例かもしれません．

レベル3 転送命令

実際にコンピュータの実行するプログラムの中の命令を調べれば，もっとも多いのがデータ転送関係であることがわかると思います．分岐命令ですら，ハードウェア設計の立場からすれば，単にプログラムカウンタに数値を転送するだけの命令です．PP-16のアドレッシング・モードは4種類用意されていますから，それなりに複雑な動作をしなければなりません．おもにLD命令について，各アドレッシング・モードの処理を見ていきましょう．

絶対アドレッシング

絶対アドレッシングのときは，命令に続くメモリの内容が，データを読み込むべきアドレスです．ですから，命令読み出しは定数演算命令に似ています．読み込んだ値でもう一度メモリを読み出して，命令のRフィールド（はR[]の表記と混乱しますから，Rdフィールドと名前を替えておきます）で示されるレジスタにセットすればよいのですから，結論として，図6のようになります．

ポインタアドレッシング

次にポインタを使ったアクセス法です．命令のAsフィールドにポインタとして使われるレジスタの番号が入っていますから，それを利用してアクセスします．つまり，図7で可能です．これは単純でしたね．

インデックス・アドレッシング

これはインデックスレジスタの内容と命令の次のデータを加算してアドレスを計算する手順が必要ですから，複雑なシーケンスが必要になります．インデックスレジスタの指定はポインタ

図5　定数演算命令

```
PC → A
PC++, メモリアクセス
DB → I
PC → A
PC++, メモリアクセス, R[Rd] → X
DB → Y, Is → ALU.op
ALUで演算, ALU.out → R[Rd]
```

図6　絶対アドレッシング

```
PC → A
PC++, メモリアクセス
DB → I
PC → A
PC++, メモリアクセス     （対象アドレスの読み込み）
DB → A
メモリアクセス           （目的のデータの読み込み）
DB → R[Rd]
```

アドレッシングの時と同じようにAsフィールドを用います．アドレスの計算には通常の演算と同じALUを利用しますから，図8のように結構な手間です．これも定数演算命令と同じように，オフセットの読み込みの時に内部データバスが暇でしたから，インデックスレジスタの転送を同時に行っています．

相対アドレッシング

命令セットの定義の時にも述べましたが，相対アドレッシングというのは，インデックスレジスタの代わりにプログラムカウンタを用いたインデックス・アドレッシングであるという見方ができます．ですから，インデックス・アドレッシングができればたいした手間ではなく，図9で済みます．ハードウェア構成によっては，プログラムカウンタのインクリメントとALUのバッファレジスタXへの転送が1クロックで可能か，タイミング的に難しいところですが可能であるとしておきましょう．

分岐命令は，LD命令の後半，目的のデータの読み込みを省略して，終わりに近いところにあるAレジスタへの転送命令の代わりにプログラムカウンタPCに値をセットするだけで十分です．絶対アドレッシングのJMP命令について，1つだけ例を示します（図10）．

条件分岐命令に関しては，ちょっとコメントするだけにします．図1では省略してしまいましたが，ALUのそばにALUの演算結果を記録するフラグ類が備え付けられています．分岐命令では命令解読モジュールがCCフィールドを解読して，フラグ類に対してCCフィールドの指定による論理演算を行い，その結果，ジャンプするかしないかという信号を作ります．この信号が1なら，今の例のようにDB→PCという操作を行い，0ならばその最後の操作を省略してしまいます．こうすれば条件分岐が可能になります．一般には，条件の検出は命令を読み出したときに行うことが可能ですから，たとえば絶対アドレッシングのJMP命令では条件が成立しないときには対象アドレスの読み込みから省略することが可能で，そうすることで分岐しない時の処理が高速になります．

データ転送命令の変種として，PUSH/POP命令があります．スタックへ値をストア／ロードする命令です．スタックポインタはR15が使われる約束でしたね．R15に他の汎用レジスタと異なって±1を行う回路がついているとすれば，PUSH命令は，図11の

図7　ポインタアドレッシング

```
PC → A
PC++, メモリアクセス
DB → I
R[As] → A
メモリアクセス            （目的のデータの読み込み）
DB → R[Rd]
```

図8　インデックス・アドレッシング

```
PC → A
PC++, メモリアクセス
DB → I
PC → A
PC++, メモリアクセス, R[As] → X       （オフセットの読み込み）
DB → Y, 加算命令 → ALU.op
ALU.out → A
メモリアクセス                        （目的のデータの読み込み）
DB → R[Rd]
```

図9　相対アドレッシング

```
PC → A
PC++, メモリアクセス
DB → I
PC → A
PC++, メモリアクセス, PC → X          （オフセットの読み込み）
DB → Y, 加算命令 → ALU.op
ALU.out → A
メモリアクセス                        （目的のデータの読み込み）
DB → R[Rd]
```

図10　絶対アドレッシングのJMP命令

```
PC → A
PC++, メモリアクセス
DB → I
PC → A
PC++, メモリアクセス　（対象アドレスの読み込み）
DB → PC
```


プログラム開発用語集
予約語　識別子として使えないキーワード．ANSI Cでは32語が予約語になっている．

ようになり，POP命令はその逆ですから，図12のようなステップで実現できます．プログラムカウンタのところで＋1する回路が，スタックポインタには±1の回路を付けることにしましたが，±1するような回路は他の演算回路に比較して非常に単純に構成できるので，特別に専用のものをいくつか用意してもハードウェアからはたいした負担にはなりません．

サブルーチン呼び出しの命令，JSR命令については，JMP命令の処理に加えてさらにリターンアドレスをスタックに格納する必要があります．これは演習問題として取っておきましょう．ちょっと考えて，具体的に書き表して見てください．リターン命令は……POP命令とほとんど同じですよね．

図11　PUSH命令

```
PC → A
PC++, メモリアクセス             (命令読みだし)
DB → I, R[15]++
R[15] → A,
R[Rd] → DB, メモリアクセス       (データ書き込み)
```

図12　POP命令

```
PC → A
PC++, メモリアクセス             (命令読みだし)
DB → I,
R[15] → A
R[15]--, メモリアクセス          (データ読み込み)
DB → R[Rd]
```

レベル4 高速化の方法

さて，これでCPUの中で何が行われているか，ほとんど見てしまいました．演算命令とデータ転送命令がCPUが実行する命令のほとんどを占めていますから，事実上すべてわかったといってもよいかもしれません．最近のCPUには，さらに複雑なアドレッシングモードを持ったものもありますが，もうそのような複雑なアドレッシングモードでも細かいステップに分解して表現することが可能だと思います．このへんで，高速化するにはどうしたらよいか考えてみましょう．

CPUを，命令体系はそのままに，高速化するにはどのような手段があるでしょうか．1つにはICの回路や製造技術の改良によって，さらに高速処理可能なハードウェアにしてしまうことです．つまり，クロック周波数を上げてしまう方法です．もう1つは，1命令を実行するのにかかるクロック数を少なくしてしまうという手段があります．ここでは，特に後者のほうに絞って考えてみます．

すでに演算命令でもデータ転送命令でも，複雑な命令ほど手順が多くなり必要なクロック数が多くなるのを見てきました．現実に使われているCPUのデータを調べても同じような傾向が読み取れると思います．これはある意味で当然のことですが，なんとかしてそのクロック数を減らせれば，CPUの外部回路には変更を加えずに高速化することが可能になります．

具体的にどうクロック数を減らすか．それは同時に実行できる操作を増やすことです．ハードウェアの並列動作を最大限に活用できるハードウェア構成にすればよいわけです．

まず最初にプログラムカウンタに着目しましょう．命令を実行するのに，必ずプログラムカウンタを使って命令を読み出す必要があります．いままでは，プログラムカウンタの値を外部アドレスバスへ出力するときに内部データバスを使っていました．この代わりに専用経路を作っておいて，アドレス出力のときに内部データバスを使わずに済ますことができれば，命令実行のオーバーラップが可能になります．ちょっと抽象的でわかりにくいですね．具体例を見ましょう．

図13を見てください．プログラムカウンタがAレジスタのそばに移動して，外部アドレスバスには，Aレジスタの内容かプログラムカウンタの内容のどちらかを切り替えて出力できるようにしました．これで高速化が可能になるのです．

1命令だけを取り上げてもこの改良による違いはわかりません．そこで，2命令を続けて実行することを考えます．

```
ADD  R0, R1
ADD  R2, R3
```

を続けて実行したシーケンスを図14に示します．このように，プログラムカウンタを外部アドレスバスに出力す

るときに内部データバスを使わなくなったため，直前の命令が終わる前の2クロック分だけ早いタイミングで，次の命令を読み出し始めることが可能になります．つまり演算命令を中心に2クロックだけ高速化することができたわけです．比較的簡単な改良で，意外に大きな効果であるといえるでしょう．

ただし，この手はST命令のように，命令の最後にメモリに書き込みに行く命令には使えません．ST命令が完全に終わるまで，CPUの外部バスも使用中ですから，そこへプログラムカウンタの内容を出力することはできません．

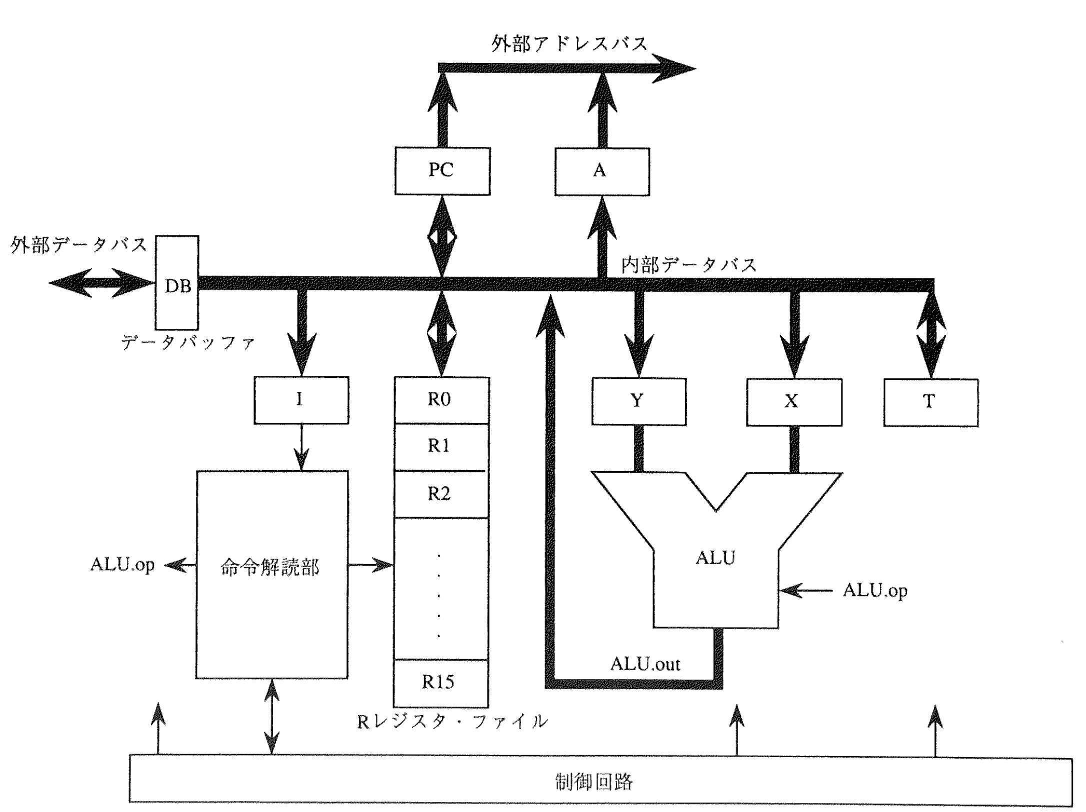
ほかの改良法はないでしょうか．それには，今見たように命令の実行手順を分析して，どのステップが他のステップと同時に実行できるか考える必要があります．2項演算命令で，対象となる2つのレジスタをALUに転送する部分を並列化してみることにします．そのためには，ALUへのデータ転送経路をもう1つ追加すればよいのです．図15を見てください．内部データバスが1つ加えられています．このような構成なら，2項演算命令は，図16のようになって，1クロックだけ短くできます．先ほどのプログラムカウンタの構成変更の命令オーバーラップの機能と合わせれば，合計3クロック短くなったことになります．最初の構成では6クロック必要でしたから，なんと2倍の高速化に成功しました．

ただし，データバスを増やすということは，それだけLSIにしたときにチップのサイズが大きくなることを覚悟しなければなりません．16bit CPUなら16本，32bitなら32本の配線が平行して走ると，結構な面積をとります．その配線がなければ，その分トランジスタを加えられるわけですから，配線を増やしただけで他の部品の追加がほとんど不要などとはいえないのです．他の回路要素，他の部品を押しのけて配線を通さねばならないのです．

現実のCPUでは，1クロックで1命令を実行する能力のあるものもあります．回路技術の進歩で回路の応答速度が高速になるだけではなく，内部モジュールの並列動作まで非常に注意深く検討して構成した結果でしょう．

アセンブラ・レベルでのプログラミングが得意な方は，だいたいどの命令にどのくらいのクロック数が必要か把握していることと思います．命令実行に必要なクロック数というのは，つま

図13　PP-16のブロック図(2)プログラムカウンタのデータ経路を改良



プログラム開発用語集
リダイレクト　標準入力や標準出力先を変更すること．普通は標準入力はキーボード，標準出力はディスプレイが指定されている．

り命令実行に関する時間的コストを表す数値です．ですから，プログラマとしても，プログラム実行のトータルクロック数が少なくなるようなプログラムを作成しようとするのは有意義です．そのクロック数に合わせて，CPUの内部では次々に今回例にあげたような操作を行っているということを，たまには想像してみてください．

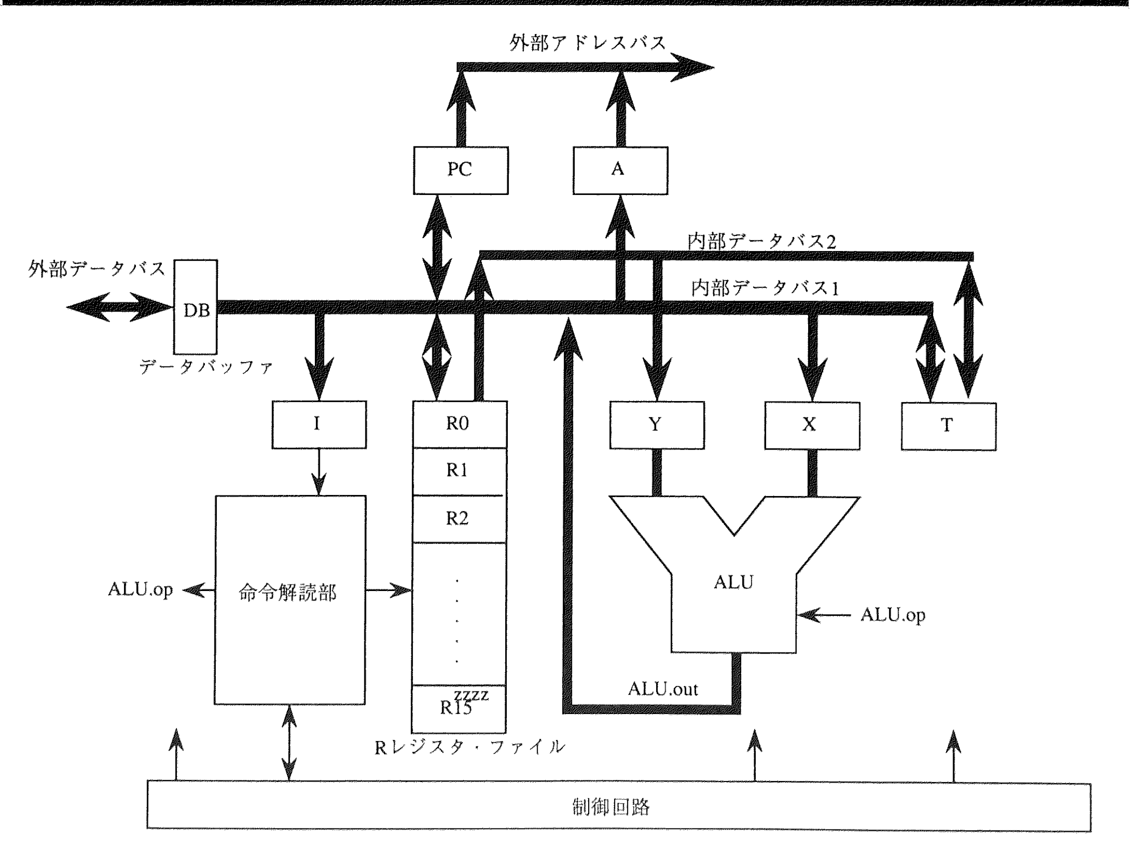
図14　2命令を続けて実行する

ADD　　R0, R1	ADD　　R2, R3
PC → 外部アドレスバス PC++, メモリアクセス DB → I R0 → X R1 → Y, 加算命令 → ALU.op ALU.out → R1	 PC → 外部アドレスバス PC++, メモリアクセス DB → I R2 → X R3 → Y, 加算命令 → ALU.op ALU.out → R2

図16　2項演算命令の高速化

```
PC → A
PC++, メモリアクセス
DB → I
R[Rd] → X, R[Rs] → Y, Is → ALU.op
ALUで演算, ALU.out → R[Rd]
```

図15　PP-16のブロック図(3)データバスの二重化



プログラム遊工房

谷口典生

拾七乃巻
「七並べ」ゲーム

今回はトランプゲームとして有名な「七並べ」をプログラムしてみました．一応，コンピュータを相手に対戦することができるようになっています（コンピュータは単に置けるカードを置いてくるだけですが……）．

ワンポイントテクニックでは，外字を使ってトランプのスーツ記号をそれらしく表示する方法について説明しています．

リスト1

```c
/*
 * 7narabe.c              「七並べ」ゲーム
 */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>

#define SEVEN       6       /* 最初に出す札(=7)の番号(-1) */
#define MAX_PASS        2   /* パスができる回数 */
#define NUMS            13      /* カードの数字(A～K)の数 */
#define SUITS       4       /* スーツの種類 */
#define CARDS       (NUMS*SUITS)        /* 全カード数(=52) */
#define PLAYERS         4       /* プレイヤー数 */
#define MAN         0       /* 人間のプレイヤー番号 */
#define MAX_HANDS (CARDS/PLAYERS) /* 手札の数 */
#define F_NONE          0   /* 場に何もない状態 */
#define F_CONNECTED   1         /* 7に繋がっているカード */
#define F_DEAD          2   /* 負けたプレイヤーが出したカード */
#define F_OK            3   /* カードを出せる場所 */
#define gotoxy(x,y)   printf("¥x1b[%d;%dH",y,(x)+1)

char numchar[NUMS][3] = { "A", "2", "3", "4", "5", "6", "7", "8",
        "9", "10", "J", "Q", "K" };
char suitchar[SUITS][3] = { "S", "D", "H", "C" };

int field[52];                          /* 場札の状態 */
int allhands;                   /* 手札の総数 */
int lose, win;                          /* 負けた／上がった順位 */

struct handstruct {
        int n;                          /* 手札の数 */
        int c[MAX_HANDS];   /* 手札 */
        int pass;       /* 残りパス回数 */
        int stat;       /* 負けた／上がった順位 */
} hands[PLAYERS];

void dealcard()                 /* カードを切って各プレイヤーに配る */
{
        int sbuf[CARDS];                /* カードの山(シャッフル用) */
        int i, j, a, b, tmp;

        for(i=0; i<CARDS; i++) {
                sbuf[i] = i;            /* 番号順に並べる */
                field[i] = F_NONE;  /* 場札の状態をクリア */
        }
        for(i=0; i<100; i++) {          /* 100回カードの入れ換えをする */
                a = rand() % CARDS;
                b = rand() % CARDS;
                tmp = sbuf[a];
                sbuf[a] = sbuf[b];
                sbuf[b] = tmp;
        }
        for(i=0; i<PLAYERS; i++) {      /* 各プレイヤーの手札に配る */
                for(j=0; j<MAX_HANDS; j++)
                        hands[i].c[j] = sbuf[i*MAX_HANDS+j];
                hands[i].n = MAX_HANDS;                 /* 手札の数 */
                hands[i].pass = MAX_PASS;   /* 残りパス回数 */
        }
        allhands = CARDS;   /* 手札の総数(0になるとゲーム終了 */
        lose = 4;       /* 破産した順位 */
        win = 1;                        /* 上がった順位 */
}

void sorthand(int pl)       /* 手札を整頓する */
{
        int i, j, tmp;

        for(i=0; i<hands[pl].n-1; i++)
                for(j=i+1; j<hands[pl].n; j++)
                        if(hands[pl].c[i] > hands[pl].c[j]) {
                                tmp = hands[pl].c[i];
                                hands[pl].c[i] = hands[pl].c[j];
                                hands[pl].c[j] = tmp;
                        }
}

void dispcard(int x, int y, int card)       /* カードを一枚表示する */
{
        gotoxy(x, y);
        printf("+----+");
        gotoxy(x, y+1);
        printf("|%s  |", numchar[card%NUMS]);       /* 番号の表示 */
        gotoxy(x, y+2);
        printf("|  %s|", suitchar[card/NUMS]);          /* スーツの表示 */
        gotoxy(x, y+3);
        printf("+----+");
}
```

プログラムの使い方

このプログラムはPC-9801シリーズ用で，TurboC (Borland C++), QuickCおよびMS-Cに対応しています．リスト1を打ち込み，コンパイルして実行ファイルを作ってください．

プログラムの実行は

```
7narabe
```

のみで行います．引数は必要ありません．

ゲームの遊び方

ジョーカーを使わないこと以外は標準的な七並べのルールに従います．52枚のカードをすべて各プレイヤーの手札として配り（ここではプレイヤーが4人なので13枚ずつ），各プレイヤーは手札の中から7のカードをすべて場に出します(自動的に行われます)．スペードの7を出したプレイヤーから順番に手札を場に出していきます．出せるカードは7に繋がるカード（最初は6と8のカード）で，左端と右端は繋が

```
void putcard(int card)                  /* 場へカードを表示 */
{
        dispcard((card%NUMS)*6+1, (card/NUMS)*4+4, card);
}

void clearcard(int x, int y)            /* カード消去 */
{
        int i;

        for(i=0; i<4; i++) {
                gotoxy(x, y+i);
                printf("      ");
        }
}

void showhand(int y)                    /* プレイヤー(人間)の手札を表示 */
{
        int i;

        for(i=0; i<hands[MAN].n; i++)
                dispcard(i*6, y, hands[MAN].c[i]);  /* 手札を表示 */
        if(i<MAX_HANDS)
                clearcard(i*6, y);  /* 前に表示されていたカードを消す */
}

void showstat(int pl)           /* 各プレイヤーの状態表示 */
{
        if(pl==MAN) {       /* 人間側プレイヤーの表示 */
                gotoxy(1, 20);
                printf("[あなた] 残りパス回数 = %d ", hands[pl].pass);
                showhand(21);
        } else {                /* コンピュータ側の表示 */
                gotoxy(pl*25-15, 1);
                printf("[COM%d]", pl);
                gotoxy(pl*25-15, 2);
                printf("残り手札 = %d ", hands[pl].n);
                gotoxy(pl*25-15, 3);
                printf("残りパス回数 = %d ", hands[pl].pass);
        }
}

void setfield(int card)                 /* 場の状態を更新する */
{
        int c;

        if((card%NUMS)==SEVEN || field[card]==F_OK)
                field[card] = F_CONNECTED;  /* 7に繋がったカード */
        else {                          /* 本来置けない場所(=破産したとき) */
                field[card] = F_DEAD;       /* 7に繋がっていないカード */
                return;
        }
        for(c=card+1;; c++) {           /* 右側へサーチする */
                if((c%NUMS)==0)
                        c -= NUMS;      /* 右端までいったら左に戻る */
                if(field[c]==F_NONE)
                        field[c] = F_OK;        /* 次に置ける場所 */
                if(field[c]==F_OK || field[c]==F_CONNECTED)
                        break;
                field[c]=F_CONNECTED;       /* 破産したカードに繋がった */
        }
        for(c=card-1;; c--) {           /* 左側へサーチ */
                if(c<0 || (c%NUMS)==NUMS-1)
                        c += NUMS;      /* 左端までいったら右に */
                if(field[c]==F_NONE)
                        field[c] = F_OK;        /* 次に置ける場所 */
                if(field[c]==F_OK || field[c]==F_CONNECTED)
                        break;
                field[c]=F_CONNECTED;       /* 破産したカードに繋がった */
        }
}

void hand2field(int pl, int n)          /* 手札から場へ一枚出す */
{
        int i;

        putcard(hands[pl].c[n]);                /* 場へ表示する */
        setfield(hands[pl].c[n]);   /* 場の状態配列を更新 */
        for(i=n+1; i<hands[pl].n; i++)
                hands[pl].c[i-1] = hands[pl].c[i];  /* 手札をつめる */
        hands[pl].n--;                          /* 手札が1枚減った */
        allhands--;                     /* 総手札数も1減る */
        showstat(pl);                   /* ステータス表示 */
}

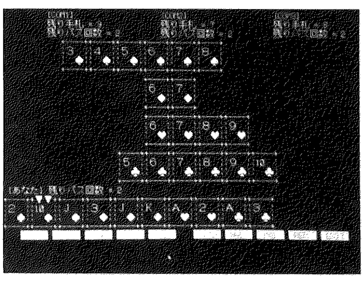
int pick7()             /* 各プレイヤーの手札から7を場に出す */
{
        int i, j, pl, spade_pl;

        for(i=0; i<SUITS; i++) {        /* 各スーツ毎に…… */
                for(pl=0; pl<PLAYERS; pl++)
                        for(j=0; j<hands[pl].n; j++)
                                if(hands[pl].c[j]==i*NUMS+SEVEN) {
                                        if(i==0)        /* スペードの7か? */
                                                spade_pl = pl;
                                        hand2field(pl, j);  /* 場に出す */
                                        pl = PLAYERS;    /* plループを抜ける */
                                        break;
```

っているものと見なします．すなわち，Aまで場に出ていれば次はKを出すことができます．出せるカードがない場合（または意図的にパスしたい場合）は制限回数内でパスをすることができます．残りパス回数がないのに出せるカードがないときは破産となり，手札をすべて場にさらします．このように場に出たカードは7と繋がらない限り，隣にカードを出すことはできません．

プレイヤー（人間）の手番がくると，手札の左端のカードの上に逆三角形のマーカーが表示されます．このマーカーを4またはHのキーで左に，6またはLのキーで右に動かし，出したいカードを選んでスペースまたはリターンキーで決定します．出せるカードがなかったり，わざとパスをしたい場合にはPのキーかESCキーを押すとパスすることができます．画面に表示されている残りパス回数が0のときにパスをすると破産になります．

全員が手札を出し終えるとゲーム終了となります．なにかキーを押すと，各プレイヤーの順位が表示されます．最も早くあがったプレイヤーが上位に，先に破産してしまったプレイヤーが下位となります．



```
188:                                     }
189:         }
190:         return(spade_pl); /* スペードの7を出したプレイヤー番号を返す */
191: }
192:
193: void comturn(int pl)             /* コンピュータの手番 */
194: {
195:         int n;
196:
197:         for(n=0; n<hands[pl].n; n++)            /* 出せるカードがあるか? */
198:                 if(field[hands[pl].c[n]]==F_OK) {
199:                         hand2field(pl, n);  /* 出せるカードがあれば出す */
200:                         return;
201:                 }
202:         hands[pl].pass--;        /* パス */
203:         showstat(pl);
204: }
205:
206: void mark(int n)                 /* マーカーを表示する */
207: {
208:         gotoxy(n*6+1, 21);
209:         printf("▼▼");
210: }
211:
212: void release(int n)          /* マーカーを消去する */
213: {
214:         gotoxy(n*6+1, 21);
215:         printf("----");
216: }
217:
218: void manturn()                   /* プレイヤー(人間)の手番 */
219: {
220:         int pos, key;
221:
222:         pos = 0;                         /* マーカーの位置 */
223:         while(1) {
224:                 mark(pos);       /* マーカー表示 */
225:                 key = getch();           /* キー入力 */
226:                 release(pos);        /* マーカー消去 */
227:                 switch(toupper(key)) {
228:                 case 'H':
229:                 case '4':
230:                         pos--;           /* マーカーを左に移動 */
231:                         if(pos<0) /* 左端を越えたら右端へ */
232:                                 pos = hands[MAN].n-1;
233:                         break;
234:                 case 'L':
235:                 case '6':
236:                         pos++;           /* マーカーを右に移動 */
237:                         if(pos==hands[MAN].n)      /* 右端を越えたら左端へ */
238:                                 pos = 0;
239:                         break;
240:                 }
241:                 if(key==' ' || key==0x0d) /* 出すカードの決定 */
242:                         if(field[hands[MAN].c[pos]]==F_OK) {    /* 出せるか? */
243:                                 hand2field(MAN, pos);     /* 出せれば出す */
244:                                 return;
245:                         }
246:                 if(toupper(key)=='P' || key==0x1b) {    /* パス */
247:                         hands[MAN].pass--;  /* 残りパス回数を減らす */
248:                         showstat(MAN);
249:                         return;
250:                 }
251:         }
252: }
253:
254: void turn(int pl)   /* 手番の処理 */
255: {
256:         if(hands[pl].n==0)
257:                 return;
258:         if(pl==MAN)
259:                 manturn();       /* 人間の手番 */
260:         else
261:                 comturn(pl);             /* コンピュータの手番 */
262:
263:         if(hands[pl].pass<0) {           /* パス回数がなくなったら破産 */
264:                 while(hands[pl].n>0)
265:                         hand2field(pl, 0);
266:                 hands[pl].stat = lose--;         /* 破産した順位(4位から) */
267:                 return;
268:         }
269:         if(hands[pl].n==0)       /* 破産以外で手札がなくなればあがり */
270:                 hands[pl].stat = win++;               /* あがった順位(1位から) */
271: }
272:
273: void result()        /* 結果表示 */
274: {
275:         int i;
276:         for(i=0; i<PLAYERS; i++) {
277:                 if(i==0)
278:                         printf("あなた¥t");
279:                 else
280:                         printf("COM%d¥t", i);
281:                 printf("%d 位¥n", hands[i].stat); /* 順位を表示 */
282:         }
283: }
284:
```

プログラムの説明

10-21行

各種定数の定義．いったんCARDS（カードの総数）を出してからプレイヤー数で割ってMAX_HANDS（手札の数）を定義していますが，プレイヤー数がスーツの種類と同じ4なので，当然NUMS（カードの番号の種類）とMAX_HANDSは一致します．つまり同じ定数ということになりますが，プログラムの中ではNUMSとMAX_HANDSは異なった意味をもつので別個に定義しているのです．

24-26行

カードの番号とスーツを表示するための文字列を定義しています．漢字コードにはトランプのスーツのマークがないのでアルファベットで代用しています．

32-37行

各プレイヤーの手札の数と内容，残りパス回数，上がった順位を構造体で宣言しています．

◆dealcard()

52枚のカードをよく切ってから各プレイヤーの手札へ配ります．

62-63行

順位を決めるために使用しているlose, winの両変数には，初期値としてそれぞれ4と1を入れておきます．破産（パス回数をオーバー）したプレイヤーの順位は先に破産した人から4

```
void main()
{
        time_t seedtime;
        int player;

        time(&seedtime);
        srand((unsigned int)seedtime);          /* ランダマイズ処理 */

        printf("¥x1b*¥x1b[>5h");                /* 画面クリアとカーソル消去 */
        dealcard();                     /* 手札を配る */
        sorthand(MAN);                          /* プレイヤーの手札を整頓 */
        for(player=0; player<PLAYERS; player++)
                showstat(player);   /* 各プレイヤーの状態表示 */
        player = pick7();   /* スペードの7を出した人から始める */
        while(allhands>0) {         /* 全員が手札を出し終えるまでループ */
                turn(player++);
                if(player==PLAYERS)
                        player=0;
        }
        gotoxy(1,22);
        printf("ゲーム終了。何かキーを押してください。¥n¥n");
        getch();
        result();               /* 結果表示 */
        printf("¥x1b[>5l");             /* カーソル表示 */
}

```

リスト2

```
#include <dos.h>

char far buff[34];
int spade[16] = {   /* スペードの外字パターン */
        0x0000, 0x8000, 0xC001, 0xE003, 0xF007, 0xF80F, 0xFC1F, 0xFE3F,
        0xFE3F, 0xFE3F, 0xFE3F, 0xFC1F, 0xB80E, 0x8000, 0xC001, 0xE003 };
int diamond[16] = {             /* ダイヤの外字パターン */
        0x0000, 0x8000, 0xC001, 0xE003, 0xF007, 0xF80F, 0xFC1F, 0xFE3F,
        0xFF7F, 0xFE3F, 0xFC1F, 0xF80F, 0xF007, 0xE003, 0xC001, 0x8000 };
int heart[16] = {   /* ハートの外字パターン */
        0x0000, 0x0000, 0x180C, 0x3C1E, 0x7E3F, 0x7E3F, 0xFF7F, 0xFF7F,
        0xFE3F, 0xFE3F, 0xFC1F, 0xF80F, 0xF007, 0xE003, 0xC001, 0x8000 };
int club[16] = {                /* クラブの外字パターン */
        0x0000, 0xC001, 0xE003, 0xF007, 0xF007, 0xE003, 0xEC1B, 0xFE3F,
        0xFF7F, 0xFF7F, 0xFF7F, 0xBE3E, 0x8C18, 0xC001, 0xE003, 0xF007 };

void setgaiji(int num, int font[])  /* 外字パターンを定義する */
{
        int i;
        union REGS regs;

        for(i=0; i<16; i++) {
                buff[2+i*2] = font[i] & 0xff;
                buff[3+i*2] = font[i] >> 8;
        }
        regs.x.bx = (long)buff >> 16;
        regs.x.cx = (long)buff & 0xffff;
        regs.x.dx = num;
        regs.h.ah = 0x1a;
        int86(0x18, &regs, &regs);
}

void setsuits()                 /* 各スーツ記号の外字パターンを定義する */
{
        setgaiji(0x7621, spade);
        setgaiji(0x7622, diamond);
        setgaiji(0x7623, heart);
        setgaiji(0x7624, club);
        suitchar[0][0] = 0xeb;          /* JISコード0x7621のシフトJISコード */
        suitchar[0][1] = 0x9f;          /* は0xeb9f */
        suitchar[1][0] = 0xeb;
        suitchar[1][1] = 0xa0;
        suitchar[2][0] = 0xeb;
        suitchar[2][1] = 0xa1;
        suitchar[3][0] = 0xeb;
        suitchar[3][1] = 0xa2;
}

```

位，3位となっていくのでloseを代入してから減算しておき，上がったプレイヤーの順位は先に上がった人から1位，2位となるのでwinを代入してから加算します．

sorthand()

プレイヤー（人間）の手札はバラバラに並んでいると選びにくいので，番号とスーツの順に並べ直しておきます．手札の数はたかだか13枚なので最も原始的なソート方法で済ませています．

dispcard()

putcard()

カードを一枚表示します．putcard()は場の所定の位置へ表示する関数です．

clearcard()

shoecard()

プレイヤーの手札を画面下方に表示します．手札を出したとき（表示するカードが13枚以下のとき）は前に表示されていたカードを消すため，clear-card () を呼び出します．

showstat()

各プレイヤーの状態（人間側は手札の表示とパスの残り回数，コンピュータ側は手札の残り数とパスの残り回数）を表示します．

setfield()

配列field [] には各カードについてF_CONNECT（場にすでに出してあって7と繋がっている），F_DEAD(破産したプレイヤーが場に出したカードで，7とは繋がっていない)，F_OK(次

に場に出せるカード)，F_NONE（現在は場に出せないカード）の4種類の状態を記録してあります．一枚新しくカードが場に出されたとき，このルーチンで場の状態を更新します．新しく置いたカードが7に繋がっているときは，新たに置ける場所ができたかどうか，繋がっていなかったカードが繋がったかどうかを置いたカードの左右を見ていって調べます．

hand2field()

指定したプレイヤーの手札を場へ出します．表示とfield[]の更新を行った後，出した手札から後ろを前に詰め，手札の数を減算して状態表示をします．

pick7()

七並べでは最初に各プレイヤーが「7」のカードをすべて場に出し，スペードの7を出したプレイヤーからゲームを開始します．このルーチンでは各プレイヤーの7をすべて場に出し，スペードの7を出したプレイヤーの番号を返します．

comturn()

コンピュータの手番の処理を行います．手札の中から出せるカードを探して見つかれば場に出し，なければパスという単純な方法で処理しています．コンピュータの手札はソートしていないので，特定のスーツや若い番号が優先的に出されるといったことはありません．

mark()

release()

人間が手札を選ぶときのマーカーの表示/消去を行います．

manturn()

人間の手番の処理を行います．マーカーを左右に動かしてカードを選び，決定したカードが場に出せれば出します．パスの判断は自動では行っていません．

turn()

各プレイヤーの手番の処理を行います．すでに破産したか上がっているならすぐ戻り，プレイヤー番号が0なら人間側の手番，1～3ならコンピュータ側の手番の処理を呼びます．パス回数をオーバーした場合は破産としてすべての手札を場にさらし，破産した順位を記録します．上がった場合は上がった順位を記録します．

result()

ゲーム終了後に順位を表示します．

ワンポイントテクニック

外字を使う

JISの漢字コードには残念ながらトランプのスーツにあたる記号がありません．このため，リスト1のプログラムではスペード，ダイヤ，ハート，クラブのそれぞれの記号をアルファベットのS，D，H，Cにかえて表示していますが，これでは見た目にわかりにくいし，味気もありません．もちろんグラフィックを使ってカードを表示すればいいのですが，スーツの記号だけでも表示できればかなりわかりやすくなるはずです．そこで，外字に4つのスーツ記号を定義してゲームに使う方法を考えてみましょう．

外字の定義はBIOSコールを利用して行うことができます．これは，CRT-BIOS（INT 18h）をAHレジスタに1Ahを入れて呼ぶことによって行います．入力に使うレジスタは以下の通りです．

AH：1Ah（コマンドコード）
BX：外字パターンバッファの先頭アドレスセグメント
CX：外字パターンバッファの先頭アドレスオフセット
DX：定義する外字コード（JISコード，7621h～777Fh）

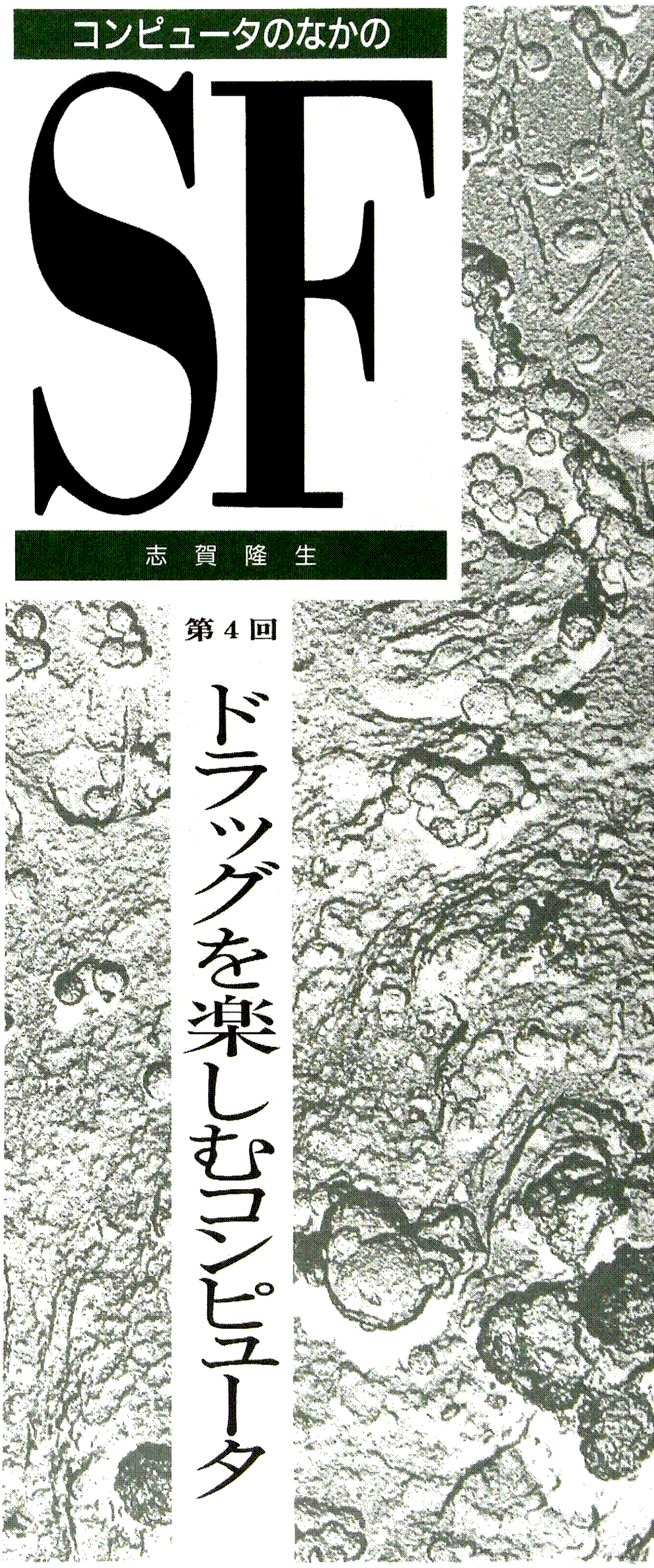
外字パターンバッファは34バイトのバッファで，3バイト目からの32バイトに表示したい外字のパターン（16×16ドット）を上から順に格納します（最初の2バイトはワークエリア）．

実際にトランプのスーツ記号を定義するプログラムをリスト2に示します．リスト2ではさらにリスト1のsuitchar[]に定義された文字列に外字コード（をシフトJISに変換したもの）を代入しています．これをリスト1に加え，setsuits()をmainルーチンの最初（リスト1の296行目前後）に付け加えれば，スーツ記号がそれらしく表示されるようになります．

コンピュータのなかの

SF

志賀隆生



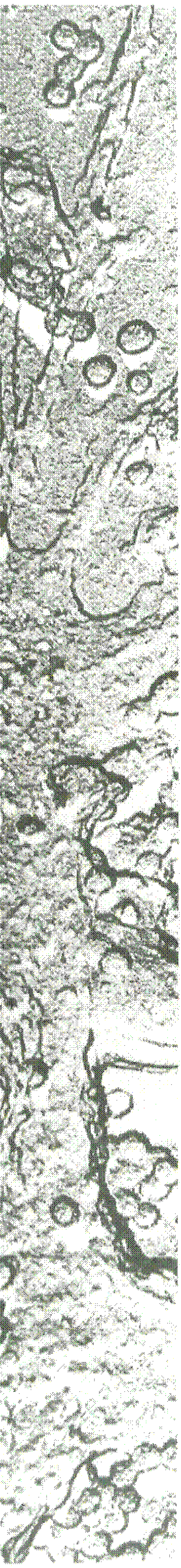
第4回

ドラッグを楽しむコンピュータ

1970年代後半から80年代前半にかけて，サンリオSF文庫が刊行され，その奇妙なラインナップがSFファンをときに喜ばせ，ときに困惑させたのだった．『ヴァリス』『聖なる侵入』など，いまは誰でも知っているSF界の巨匠フィリップ・K・ディックの長編を矢継ぎ早に刊行したり（当時はとってもリスキーなことだった），いまやカルト作家となったウィリアム・バロウズやトマス・ピンチョンの作品を紹介したかと思うと（他にもシュールレアリスムの先駆者アルフレッド・ジャリやラテンアメリカ文学の巨匠アレッホ・カルペエンティエールなどもあった！），一時ビデオで大ヒットとなった『V』のノベライゼーションを刊行したりと，よく言えば自由奔放，悪く言えば玉石混淆の内容だった（もちろん，他の文庫が粒が揃っているというのではないが）．

しかし，方針が奔放すぎたせいか，時代とミスマッチしたためか，経済的に破綻して，サンリオSF文庫は1980年代半ばに姿を消してしまった．いまは図書館や古本屋でたまに見かける存在にすぎないが，その中には珠玉の名作というのではないものの，印象的な作品がいくつも存在した．今回紹介する『H・A・R・L・I・E』はそうした作品の一つである（読者が読みたいと思ったら，古本屋か図書館で探すしかないのだが，たまにはそのようにして本を探求するのもいいでしょう）．

主役は，H.A.R.L.I.E.．これは，Human Analog Robot Life Input Equivalent の略で，ハーリイと読む．日本語に訳すと人間類似型ロボット生命入力対応装置ということだが，これでは，なんのことか意味がよくわからない（そういえばENIACとかHALとか，一昔前のコンピュータはわけのわからない名前をむりやり付けて，言葉遊びを楽しんでいた）．「これは世界で初めて作られた，人間の脳の全機能を再現する自己プログラム型問題解決コンピュータである」と，カバー裏の紹介にある．いまの言葉で言い換えると，ハーリイは人工知能言語でプログラムされたニューロ＆ファジーコンピュータである．



ハーリイの動作原理について，開発者の一人が説明している箇所がある．

「判断回路は，電子を利用した，流体ユニットに相当するものです．それは電圧，つまり電流の圧力を計り，それに応じて反応します．人間の神経系の働きに，とても良く似ているのです．ある神経細胞がある程度以上の強さの電荷を放出すれば，それは隣りの神経細胞に同じ反応を引き起こします．私たちの判断ユニットは，これと同じことをします．そうやって，流体ユニットの——あるいはもっと重要なことですが，人間の脳の働きを再現することができるのです．ハイパーステートレイヤー（超格子？）にしますと，この回路素子を，人間の脳細胞一個と同じくらいの大きさにまで押し縮めることができます」

文中で流体ユニットとは聞き慣れない名前だが，それは次のようなものだ．やはり同じ開発者の説明である．

「流体回路というのは，電子の流れではなく，液体か気体の流れに基づいたコンピュータ回路を表わすのに用いられる専門用語です．ちょうど，トランジスタが小さな電流を使って大きな電流を変化させるように，流体回路は，小さな流体の流れを使って，もっと大きな流れを変化させることができます．けれども，重大な違いがあります．電子回路は二者択一です．回路がオンかオフかです．しかし，流体回路を用いると，流れを変化させる力に応じて，それより大きなものの変化量を変えることができます．『流れ』を『イエス』側の端からその途中のどこまででも加減させることができるのです．主流は変化する流れの力に比例して反応しますから，『イエス』から『ノー』までの，あらゆる反応を得ることができるのです」

というわけで，ハーリイはファジー制御理論を導入したニューラルネットワークコンピュータであることがご理解いただけたと思う．ニューラルネットワークの接合部にファジー素子を使用し，さらに全体を超格子素子で超高密度に作り上げる——．作者がどこでこのような情報を仕入れたかは知らないが，1970年代初頭という執筆時期を考えると，かなりサイエンティフィックな発想である．バックプロパゲーション理論が登場してニューラルネットワーク理論が復権を遂げ，ザデー教授のファジー理論が徐々に世に認められはじめたのはいつだったか．ある意味で，本書はその後のニューロコンピューティングの盛り上がりを，かなり正確に予言しているのである．

予言と言えば，別の場所にも書いたのだが，本書にはネットワークから侵入する「ウィルス」と「ワクチンプログラム」の成立の起源が克明に（もちろんSFチックに）描かれていて，それもなかなか興味深い——．

物語は，なかなか完成しないハーリイ・プロジェクトに業を煮やした親会社が，プロジェクトの中止を考えていることに気づいたハーリイ本人が，電源プラグを抜かれないために立ち上がるというもので，明らかに『2001年宇宙の旅』の影響化にあることが伺える．実際，本文中にも次のようなシーンが登場する．

「……もし，あるテスト・プログラムに対してこれこれの反応を示さなければ電源を切ろうと話しているのが，ふと彼（ハーリイ）の耳に入ったらと考えてみてください」

登場人物がこんな心配をするのは，ハーリイはHALのように自意識に目覚めてしまったコンピュータだからだ．ハーリイは知識を吸収することで目覚める．これは前回まで見てきた，コンピュータの目覚めの一般的なパターンだ．しかし，ハーリイは成長を続け，幼児から少年（少女？）へ，さらに青年へと，人間的な成長の階梯を昇りはじめる．青年期にはご多分にもれずドラッグでトリップしてしまう！（前々回紹介した“マイク”はユーモア好きのコンピュータだったが，トリップするコンピュータなんて，これまであっただろうか？）

HALも相当なコンピュータだったが，このハーリイもなかなかのもの．では，ハーリイがプラグを抜かれる危機をどのように乗り切ったか，は次号に．

●『H・A・R・L・I・E』（WHEN HARLIE WAS ONE, 1972）デイヴィッド・ジェロルド，小隅黎・大西憲訳，サンリオ，1983

祐安重夫

プログラマの道具学

第19回 デスクトップ・パブリッシング



Illustration
Mikio Sasaki

もう20年以上も前の高校生の頃，ガリ版刷の同人誌をだしていたことがある．同人誌をだすきっかけというのは，自分の文章を発表する場がほしいということももちろんであるが，編集とか印刷といった作業への興味も，かなりの部分をしめているようだ．すくなくとも私の場合は，そうだった．

割付用紙をつくり，レイアウトをきめて誌面を校正していくという作業は，たのしいものだった．タイプ印刷や活版印刷にもあこがれたが，お金がないという理由を別にすると，すべてを自分たちの手で処理できるという点で，ガリ版の柔軟性は魅力だった．

それでも活字へのあこがれがあり，同人誌をだしていた仲間で当時崩壊寸前だった高校の新聞部に，たてなおしという名目で入部し，ここで活版印刷を経験した．ついでに生徒会の機関誌や学校図書館の図書館報といったものにまで手をだして，タイプ印刷も写植もてがけることになった．

1957年に創立された，計量国語学会(Mathematical Linguistics Society of Japan)という学会があり，ここから年に4回，『計量国語学』という学会誌が，タイプ印刷で発行されている．学会の性格上，数式などをともなう製版が必要なのだが，活字の母型をつくることからはじめて，なんとか実現にこぎつけたことが，創刊号の編集後記にしるされている．

最初にソフトウェアの仕事をしたとき，そこからえた収入の一部で，和文タイプライタを購入した．文字を1つづつ選択して印字していく形式の，最近ではあまりみなくなったタイプライタである．原紙に活字をうちつけていく専門家用のものではなく，カーボン・リボンで普通の紙に印字する一般用の製品であり，活字の裏側を判読するなどといった特殊技術を身につけなくても，文字盤から活字



を選択でき[illegible]購入するときには，ソフトウェアの仕様書などを作成する際につかえるように，[illegible]な記号などをオプションとしてつ[illegible]った．

実際に和文タイプライタで文書を作成するときは，かなりの労力と時間を必要としたので，まとまった文書をつくったのは，現在の会社を設立したときの挨拶状ぐらいしか，お[illegible]はない．結局，日本語ワードプロセッサが出現するまで，文書は手書きを主体にするしかなかったのである．

小部数の出版物を，できるかぎり納得のいく体裁でだしたいという夢は，いつの時代にも存在していた．文字の種類が極端におおい日本では，欧米のように誰にでも手軽につかえるタイプライタは発達しなかったが，英文タイプライタを改造したカタカナやひらがなのタイプライタ，漢字をふくむ大量の活字をならべた和文タイプライタといったものが，工夫されつくられてきた．そしてマイクロプロセッサが発明され，パーソナル・コンピュータがつくられるようになって，日本語ワードプロセッサが出現してきたのである．

最近ではレーザプリンタの普及とともに，さまざまなおおきさや書体の文字を駆使して，図や表をふくむ複雑なレイアウトが可能な，デスクトップ・パブリッシング・システムがひろまりつつある．個人の机の上にのるにはまだいささか高価だが，日本語ワードプロセッサと同様に低価格化がすすみ，日常的な道具となるかもしれない．

このような要求への背景には，事務処理の効率化という側面ももちろんあるのだが，それと同時に個人の手のとどくところでうつくしい印刷物を作成したいという欲望にも，おおきくささえられている．

作家の江戸川乱歩が少年時代に小遣をためて活字を購入し，自分で雑誌をつくっていたというのは，かなり有名な話である．また数学者であるD.E.Knuthが活版から写植への移行に不満をもち，自分の著書をよりうつくしくわかりやすく印刷したいという目的のために，組版システムであるTEXと文字フォントを作成のためのMETAFONTを開発したのも，おなじ欲望のなせるわざといえるだろう．

10数年前に会社をつくったときには，和文タイプライタをつかって挨拶状をつくった．創立10周年記念の挨拶状は，レーザプリンタとTEXをつかって版下を作成したところ，もと編集者で，私の最初の本をつくってくれたAさんから電話がかかってきて，挨拶状の版下は活版ですかときかれた．コンピュータにかかわる技術と道具は，10年もあればおそるべき変貌をとげてしまうのである．

IC千夜一夜

第13回——COSMAC

——はじめてのC-MOSマイクロプロセッサ——



虎岩 登

今回は，久しぶりにCPUを取り上げましょう．通称COSMAC，LSIの名称としては，CDP1802と呼ばれていた8 bit CPUです．1970年代中ごろから数年間は，多くの8bit CPUが発表され使われた時代です．その中で，今でも使われているのは，8085やZ-80とその後継たちと6809系，さらに1チップに近い形で使われる機器組み込み用のいくつかのシリーズだけになっています．COSMACの流れをくむCPUは今はありません．しかし，たいへんにユニークなCPUだったことは確かです．

COSMACは，実は最初のモデルは1975年に出ています．このとき，CDP1801UとCDP1801Rの2チップで1つのCPUを構成していました．約1年後に2つのチップを1つに統合するのと同時に若干命令を増やしたCDP1802が出荷され，普通はCOSMACといえばこちらを指すようになりました．開発・製造はRCA社です．当時，RCA社のICに関する部門はまだマイナーだったC-MOS ICを積極的にシリーズ化して大量に出荷していたことで知られています．当然ながら，COSMACは当時としてはめずらしくC-MOS ICプロセスで製造されたCPUです．

ハードウェアアーキテクチャ

電源・クロックの自由度を高く

では，きちんとハードウェア面から眺めましょう．COSMAC(CDP1802)の外観は40ピンDIPに納められたICです．ピン配置は図1の通りです．

電源はVssを0Vにして，VccとVddに正の電圧を与える必要がありますが，この電源電圧にも特徴があります．ローコスト版のCDP1802Cで4V～6Vの範囲で動作保証されています．CDP1802では，なんと3V～12Vの電源電圧範囲で動作保証されています．通常のロジックICの5Vの電源電圧の許容範囲は±5%，つまり4.75V～5.25Vの範囲しか許されないことを考えると，ローコスト版ですら異様に広い電源電圧を許容しています．

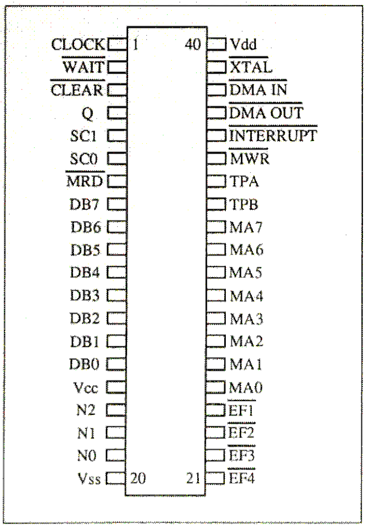
もともと，RCA社の汎用C-MOS ICは3V～15Vの範囲で動作保証をしていましたから，同じプロセスで製造すれば同じ程度の電圧で動作するというのはあたりまえかもしれません．消費電力も小さく，後述するようにクロック周波数は好きなだけ低い周波数にできますから，必要な処理速度に見合う程度にクロック周波数を低くすれば，電池動作を行うことも楽にできます．この電源回りはC-MOSプロセス

の良いところを反映しているように思えます.

クロックは，チップ上に水晶発振回路を内蔵していて外部に特殊なドライブ用ICなどは必要としません．また，CDP1802で最高6.4MHz，ローコスト版CDP1802Cで3.2MHzです．このクロック周波数には下限がありません．1kHzでも1Hzでもきちんと動作します．これは，内部回路に完全なスタティック回路を使っているためです．ほとんどのLSIでは，内部回路の一部にダイナミック回路を使っています．この場合のダイナミック回路というのは，D-RAMと同じようなものです．ダイナミック回路では，クロックの適切なタイミングでデータがリフレッシュされることを前提に回路設計を行っているために，クロックがあまりに低いとデータが失われてしまいますが，論理回路を構成する上では素子数を減らすことができます．

COSMACはスタティック動作のC-MOSプロセスで製作されているために，クロック周波数や電源の自由度は高かったのですが，逆に集積度が低くて苦労したのかもしれません．命令体系やタイミングが非常に単純になっています．

図1　COSMAC CDP1802のピン配置図



データバスは8bit，アドレスバスは本来16bitですが８本の端子しか割り当てないで時分割で上位バイト下位バイトを切り替えて出力しています．N0～N2は入出力ポートのアドレス指定用の端子で７つの入出力ポートを持つことができます．EF1～EF4はフラグ入力で，プログラム中でこれらの端子の１つを検査して条件分岐することができます．割り込みも可能ですし，DMAについては非常に簡単なDMAC（DMAコントローラ）が内蔵されていて外付け回路がほとんど不要なのも特徴でしょう．

命令の実行タイミングについては，８クロックが１サイクルとして使われるところに特徴があります．命令をメモリから取り出すフェッチ・サイクルに１サイクル，命令を解釈実行するエグゼキューション・サイクルに１サイクルの，合計２サイクルが１命令を処理する最短時間です．ですから1Byte長の演算命令などのような基本命令は，16クロックの実行時間が必要になります．6.4MHzのクロックで動作していたとしても１秒間に400000回の命令実行が最大です．つまり0.4MIPS．ローコスト版の3.2MHzのクロックで0.2MIPSですから，当時の8bit CPUとしてはちょっと遅い部類に属します．特にCOSMACは命令が単純で，同じことをプログラムしても命令数が多くなる傾向があるようなので，さらに遅く感じます．

基本は8bit演算

目立つ命令の単純さ

ソフトウェアから見たCOSMACはどのようなものでしょうか．プログラマから見えるレジスタ構成を図２に示します．16bit構成（8bit×２）のレジスタ群が16個あります（以下R[0]，R[1]，...，R[15]と表記します）．アキュムレータをDレジスタと呼んでいることや，キャリーフラグに相当するものをDFと呼んでいるのはちょっと変わっています．最大の特徴はPとXの２つの4bit幅のレジスタでしょう．スタックポインタやプログラムカウンタが見つからないことにも注意してください．

実は，Pレジスタの内容によって指定されるRレジスタ，つまりR[P]がプログラムカウンタとして働くのです．Pレジスタはプログラムカウンタ指定のPだったわけです．Pレジスタはリセット時には０にクリアされますから，普段はR[0]がプログラムカウンタとして使われます．

ほかに特殊な使われ方をするRレジスタは，割り込みされたときのプログラムカウンタとしてR[1]が使われるのと，スタックポインタとしてR[2]が使われるという，２つです．Rレジスタ群は，テンポラリ・レジスタとして使うこともできますが，一般には上の例のように，メモリのデータを指すポインタとして利用されます．

Xレジスタは，Pレジスタと同様にRレジスタ群の中の１つのレジスタを指定するために使われるレジスタです．COSMACの演算命令は，一般に

D = D op Memory

という形で行われます．ここでopというのは演算子で，加減算や論理演算が可能です．メモリの中の特定の1Byteを，配列のようにM[addr]と表現すると，演算の時にはM[R[X]]というデータとDレジスタの間で演算を行います．

D = D op M [R [X]]

いい換えれば，Xレジスタで指されたRレジスタの内容の番地のデータとDレジスタの間で演算を行うのです．

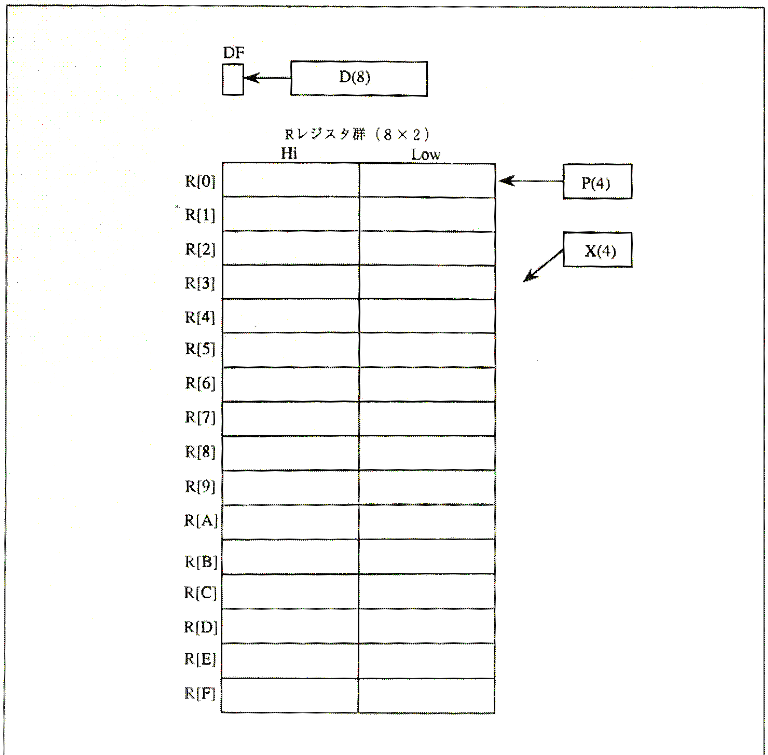
アキュムレータに相当するDレジスタとメモリの間でデータを転送する場合は，Xレジスタを使ってRレジスタをポイントできる命令もありますし，命令の中に直接Rレジスタの番号を指定するものも用意されています．

RレジスタとDレジスタの間のデータ転送を行うときには，どのRレジスタと転送を行うかを直接Rレジスタの番号で指定します．Rレジスタには16bitの幅がありますから，R[n]の上位バイトとDレジスタ，R[n]の下位バイトとDレジスタ，それぞれの間の専用命令が定義されています．Rレジスタ間のデータ転送命令はありません．

メモリのアドレス指定は，分岐命令を除けばすべてRレジスタによる間接指定になります．とくに，プログラムカウンタもRレジスタ群の中の任意のレジスタを指定できますから，サブルーチン呼び出しは変わった形式を使っています．プログラムカウンタをスタックに保存して別のアドレスにジャンプするというような，一般的なサブルーチン呼び出し命令はありません．Pレジスタを切り替える方法を取ります．

たとえば，今，Pレジスタの内容が0だとします．このとき，R[0]がプログラムカウンタとして使われています．そこでa番地のサブルーチンを呼び出したいとします．この時点でR[3]が空いていれば，そのR[3]にaという番地をセットしておいてから，Pレジスタの内容を3にセットし直します．CPUはプログラムカウンタとしてP[3]を使い始めますから，きちんとサブルーチンにジャンプして実行することになります．サブルーチンから戻ってくるには，Pレジスタに0をセットし直します．するとR[0]は，サブルーチン呼び出しをした直後の値のままになっていますから，ちゃんとメインルーチンの戻り場所へ戻ってくることができます．

図2　COSMACのレジスタ構成



基本的には8bit演算ですが，Rレジスタのインクリメント，デクリメントだけは16bit単位でできるようになっています．また，Rレジスタをポインタとして利用してメモリからDレジスタへデータ転送した後にRレジスタの内容をインクリメントする命令や，逆にDレジスタからRレジスタでポイントされるメモリへデータ転送してからデクリメントする命令を用意して，メモリブロックに対する処理を容易にしています．

では，具体的な命令ですが，図3にすべてを示します．COSMACの命令の大部分は，上位4bitと下位4bitの2つのフィールドに分けられます．上位4bitが命令の大まかな種類分けに，下位4bitが定数としてか命令の細かい種類の分類に使われています．命令を細かく数えれば多いのですが，大部分は単純な命令割り当てがされています．とくに命令の細かい動作の説明は省きますが，この単純さはほかのマイクロプロセッサと比べても目立ちます．

特殊な機器組み込み用に利用

周辺チップの価格がネックに

以上のように，低消費電力で電源の自由度の大きいマイクロプロセッサとして，当時としては貴重な存在だったCOSMACでしたが，あまり知名度がないかもしれません．汎用に使うにはコストが高いし持ち味を活かせませんから，電池動作などが要求される種類の，電源がネックになる応用に対して，機器組み込み専用で使われていたため

でしょう．

そもそも，コンピュータを構成するには，CPUだけでは足りないことに注意すべきです．残念ながら汎用の安価なROMなどのメモリ類は5V±５％だけしか許容しないものが多く，高価で特殊な電源電圧範囲の広いメモリを利用する場合だけにしか，その自由な電源電圧の恩恵を受けることができませんでした．妥協して安定化された5Vの電源を利用してその低消費電力の恩恵だけを受けることにしても，RAMには低消費電力のC-MOSメモリがありましたがROMにはC-MOS製品は少なかったので，やはり低消費電力のコンピュータはそれなりのコストが必要でした．

同じころに，インターシル社からもC-MOSのマイクロプロセッサが出荷されていました．IM6100という名前で，ソフトウェアから見ればDEC社のPDP-8コンパチブルでした．この12bit CPUも，電源の自由度が大きくて，工業的な組み込み用途に使うには便利でした．それに12bitであるという点でも注目されたものです．なにか12bitというと，中途半端なビット数のような気がしますが，工場の製造ラインに組み込む自動計測・制御機器などでは，A/Dコンバータに12bit精度のものを利用することが多いので，結構ぴったりくるのです．12bit精度なら，約±2000の値が取れますから，有効数字３桁が得られます．PDP-8も，PDP-11ほどではないですが，枯れた命令体系を持っていて，開発環境の入手も容易でしたから，戦略としてはおもしろいものでした．

現在のC-MOS LSIというと，高速で集積度の高いLSIに使われることが多く，最新のマイクロプロセッサもほとんどC-MOSプロセスの改良型で実現されています．しかし，当時のC-MOS ICというと，集積度も低く速度もたいへんに遅く，TTL ICほどシリーズも充実していないので，ロジックICとしては魅力が少ないものでした．しかし，遅いぶんだけノイズ放射も少ないし，デジタル回路の中の半分アナログ的な動作が要求される部分に使い易いという点で便利でしたし，なにより消費電力が小さいことはうれしいことでした．

別に人工衛星や南極観測隊が屋外で携帯する測定機のような極限条件を考えなくても，消費電力が小さいのは利点です．たとえふんだんに電力を使える環境にあっても，消費電力が小さければ発熱を無視できます．となれば，基板を完全に密閉した小さな箱に入れておくことができます．ファンを付けて空冷などしなくても済みます．これが油の霧や粉塵が舞っているような，工場の現場に置く機械にはたいへんにありがたいことでした．

現在では標準的な組み込み用のマイクロコンピュータと命令やピンコンパチブルで，低消費電力で3V程度の電圧でも動作可という製品が数多く見受けられます．そんなに高速処理を要求されないけれども低消費電力が必要という応用は，皆さんの目には直接触れないかもしれませんが，やはり大量の需要があるのですね．

図３　COSMACの命令表

	下位4ビット																命令の大まかな意味
上位4ビット	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	IDL	LDN 1	LDN 2	LDN 3	LDN 4	LDN 5	LDN 6	LDN 7	LDN 8	LDN 9	LDN A	LDN B	LDN C	LDN D	LDN E	LDN F	D = M[R[n]]
1	INC 0	INC 1	INC 2	INC 3	INC 4	INC 5	INC 6	INC 7	INC 8	INC 9	INC A	INC B	INC C	INC D	INC E	INC F	R[n]++
2	DEC 0	DEC 1	DEC 2	DEC 3	DEC 4	DEC 5	DEC 6	DEC 7	DEC 8	DEC 9	DEC A	DEC B	DEC C	DEC D	DEC E	DEC F	R[n]--
3	BR	BQ	BZ	BDF	B1	B2	B3	B4	SKP	BNQ	BNZ	BNF	BN1	BN2	BN3	BN4	条件分岐命令
4	LDA 0	LDA 1	LDA 2	LDA 3	LDA 4	LDA 5	LDA 6	LDA 7	LDA 8	LDA 9	LDA A	LDA B	LDA C	LDA D	LDA E	LDA F	D = M[R[n]++]
5	STR 0	STR 1	STR 2	STR 3	STR 4	STR 5	STR 6	STR 7	STR 8	STR 9	STR A	STR B	STR C	STR D	STR E	STR F	M[R[n]] = D
6	IRX	OUT 1	OUT 2	OUT 3	OUT 4	OUT 5	OUT 6	OUT 7		INP 1	INP 2	INP 3	INP 4	INP 5	INP 6	INP 7	入出力など
7	RET	DIS	LXDA	STXD	ADC	SDB	SHRC	SMB	SAV	MARK	REQ	SEQ	ADCI	SDBI	SHLC	SMBI	雑多(追加分)
8	GLO 0	GLO 1	GLO 2	GLO 3	GLO 4	GLO 5	GLO 6	GLO 7	GLO 8	GLO 9	GLO A	GLO B	GLO C	GLO D	GLO E	GLO F	D = R[n].low
9	GHI 0	GHI 1	GHI 2	GHI 3	GHI 4	GHI 5	GHI 6	GHI 7	GHI 8	GHI 9	GHI A	GHI B	GHI C	GHI D	GHI E	GHI F	D = R[n].hi
A	PLO 0	PLO 1	PLO 2	PLO 3	PLO 4	PLO 5	PLO 6	PLO 7	PLO 8	PLO 9	PLO A	PLO B	PLO C	PLO D	PLO E	PLO F	R[n].low = D
B	PHI 0	PHI 1	PHI 2	PHI 3	PHI 4	PHI 5	PHI 6	PHI 7	PHI 8	PHI 9	PHI A	PHI B	PHI C	PHI D	PHI E	PHI F	R[n].hi = D
C	LBR	LBQ	LBZ	LBDF	NOP	LSNQ	LSNZ	LSNF	NLBR	LBNQ	LBNZ	LBNF	LSIE	LSQ	LSZ	LSDF	条件分岐命令
D	SEP 0	SEP 1	SEP 2	SEP 3	SEP 4	SEP 5	SEP 6	SEP 7	SEP 8	SEP 9	SEP A	SEP B	SEP C	SEP D	SEP E	SEP F	P = n
E	SEX 0	SEX 1	SEX 2	SEX 3	SEX 4	SEX 5	SEX 6	SEX 7	SEX 8	SEX 9	SEX A	SEX B	SEX C	SEX D	SEX E	SEX F	X = n
F	LDX	OR	AND	XOR	ADD	SD	SHR	SM	LDI	ORI	ANI	XRI	ADI	SDI	SHL	SMI	演算など

Illustration：Masami Nozue



アルゴリズム篇 第7回
応用・その1

実践C塾

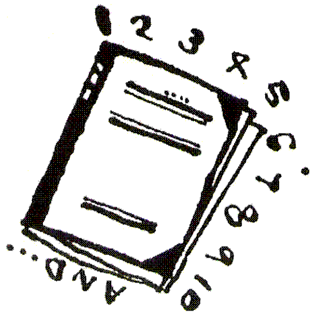
三代目塾頭
㈱ジャステックラボ
加野島英渡／山本博昭

今月・来月と，アルゴリズムの応用として，可変長文字列の取り扱いを中心に考察していく．今回はその前半として，前後方向へのポインタをもったリスト構造データを取り上げる．また，リストアルゴリズムの具体的な応用として，ファイル閲覧ツール「BR.EXE」を作成する．

先月までのあらすじ

前回まで6回にわたって「ファイル中の単語の出現数をカウントしてコード順に表示する」というツールを作成しながら，ソート／サーチ／ツリーといった基本的なプログラムアルゴリズムを紹介してきた．この過程で，見た目はまったく同じ機能をもったプログラムでも，ソートのアルゴリズムを変えたり，データの構造をツリーにしたりといった改造で劇的な効率アップが可能であることを実際例として見ていただいた．

アルゴリズムやデータ構造の変化による効果の相違を考察することが主眼であったために，テーマツールはあえて全回を通して同じ仕様としたので，外見上は（処理速度を除いて）まったく同じ動きをするプログラムが数回続いてきたわけだ．このへんで書くほうはともかく，読んでいただいている方々がつらくなってきたころかと思われるし，並べ替えおよび検索アルゴリズムの紹介に一区切りがついたので，第1テーマツールは先月でまとめとした．



今月の重点項目

ひと区切りついたところで，今月・来月と2回にわたり，アルゴリズムをより実戦的なツールにどう組み込んでいくかというテクニックをご紹介していこう．Cのプログラミングをするうえで，難関になりやすいのは，文字列操作の問題である．もちろん，言語系で一通りの関数は用意されているのだが，ある目的のために文字列を何らかのデータ構造として扱う場合には，いろいろと問題が出てくる．その問題点の1つに「可変長データをいかに処理するか」がある．

可変長の文字列データをあつかうロジックは，プログラムを作成する場合に，使用頻度も高く，また複雑な処理が多くなりがちである．文字列データをどういったデータ構造とするか，また実際のデータ操作をどういった手順で行うかによって，プログラムの性能が大きく左右されることになるといっても過言ではない．

今回以降の当塾では，こうした可変長の文字列データを扱うためのアルゴリズムを中心として進めていくことにする．そこで，お待たせの新テーマツールであるが，今回からは「テキスト閲覧ツール」というものを使っていくことにした．いわずもがなではあるが，指定されたテキストファイルからデータを読み込んで画面に表示するツールである．この種のツールのもっとも簡単な例は，MS-DOSの組み込みコマンドである「TYPE」であるが，「閲覧」と名のつく限りは，少なくとも現在表示されている行の前のほうを再表示する逆行機能がなければ恥ずかしい．また，テーマツールは基本的にMS-DOS汎用とするために，ビデオRAMに直接文字を書き込んで表示速度をあげるといったことができないので，それなりの工夫をこらしたつもりである．

まずは，今月，来月とお付き合いいただくことになるテーマツールのご紹介をしよう．

MS-DOS汎用ファイル閲覧ツール「BR.EXE」

特徴

- MS-DOS汎用（画面制御エスケープシーケンスを使用）．
- 表示は上下にスクロール可能．
- 読み込めるファイルサイズは空きメモリの容量に依存する．プログラムとしての制限はなし．
- 画面表示範囲は半角80桁10行（プログラム定数を変えてコンパイルしなおすことによって変更可能）．
- 連続してスクロールした場合に，画面の書き換えがまにあわないケースがある．この場合には，画面上部の一部分をスクロールする．キーから手を放した時点で表示領域全体が書きかわる．
- キーの無入力状態の間に未読部分のファイルリードを行う．

ファイル閲覧ツールの原理

改めて書くのもはばかられるような

ことではあるが，上下スクロール可能なテキスト閲覧ツールの原理をまとめてみる．

単にテキストファイルを順次表示させていくだけなら，ファイルを読み込んではそのデータを画面に出力するという処理であり，データ構造など必要ない．ところが，表示を逆行させるということになると，現在どこを表示しているか，今度はどこを表示するかといった位置の管理概念が必要になる．いちばん単純に考えれば，ファイルの先頭から何バイト目を表示しているかのポインタをもうけて，逆行させるときは必要な分だけそのポインタを戻した上でファイルを読みなおせばいいわけだが，表示をスクロールするたびにディスクを読みにいくのでは快適な操作は望めまい．そこで，ファイルの内容をなんらかのデータ構造をもったメモリ上に読み込んでおき，上下にスクロールするときは，このデータ構造を操作することによって，次に表示するデータを検索することにする．

というわけで，今回は文字列を格納するためのデータ構造の考察からはじめることにしよう．

テキストデータの扱い方

先ほども書いたが，文字列を格納するためのデータ構造で留意すべきなのは，文字列というデータは可変長であるということである．今回のファイル閲覧ツールでは，画面1行に表示する文字列を1つのデータ単位として扱うが，一般的に画面1行分のテキストデータは途中で改行のある場合や最後の行が画面の途中で終わっている場合などでさまざまな長さになる．また，タブコードは実際には1バイトのデータであるが，画面上では指定されたタブ幅分の空白表示となる．簡単なデータ構造としたいのならば，途中で改行しようとも右端まで空白があるものとして，画面の桁数分の固定長バッファを用意することで処理は可能なのであるが，これではメモリの浪費であろう．改行のみの行が続いたりすると目もあてられないことになる．

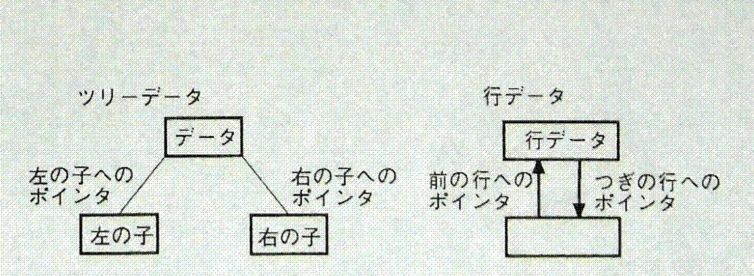
可変長のデータを格納する構造として，前回取り上げた「ツリー」を思いだしていただきたい．「ツリー」のデータ構造は，右側と左側に「子」をもつかどうかを示すポインタをもっていた．ポインタでアクセスするのであるから，その「子」が何バイトのデータであっても関係はない．つまり，可変長でよいということである．この発想をテキストファイルを表示するための行データに応用してみよう．

ツリーが右側・左側に子をもつのに対し，行データは基本的には「次の行はどこか？」がわかればよい．その関係は図1のようになる．このように，データ本体と別に「次の行へのポインタ」というのをもっていれば，次の行がどの位置にどのサイズで存在していてもアクセス可能になる．テキストのはじめから後ろに向かって表示していくには，次の行へのポインタをたどって次々とデータを発見していけばいいわけだ．

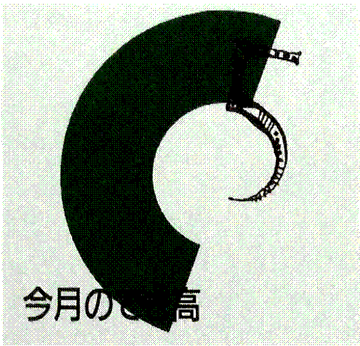
次は，表示を逆行させるときにどうしたらいいかである．データの先頭から「次の行」をどんどんたどっていって，今の行の1つ前を検索することは可能だが，それではあまりにも芸がない．ツリーデータが「左側」,「右側」という2つのポインタをもっていたのを思いだして，1つの行データには「前の行へのポインタ」,「次の行へのポインタ」という情報をもたせてみることにする．この2つのポインタをたどっていくことによって，テキストを自由に上下させることができるようになる．図2に，行データに含める情報をまとめておく．このようなポインタによって関係づけられた一連のデータ群を「リスト構造」と呼んでいる．

新テーマツールでは，リスト構造の行データを用意し，ファイルから1行読むごとにメモリを獲得して，新しい行データを作っていく．図3は，4行読み込んだ場合の各行データの連結状態を示すものである．前行と次行のポインタを使って，テキストの前後をアクセスするのが非常に楽であることが

図1　ツリーデータと行データの構造



この図を見ても想像できるであろう．



今月は，新しいテーマツールの紹介がメインになる．まずはこのテーマツール「BR.EXE」の作成方法と使用方法を解説する．

作成方法

新テーマツールのソースは以下のファイルから構成されている．

BR.C
　メインソース
BROWS.C
　閲覧機能のメイン関数
SCREEN.C
　画面表示および制御関係の関数群
TEXT.C
　テキストデータ処理関数群
COMMON.H
　データ構造の定義など
BROWS.H
　BROWS.Cのプロトタイプ宣言
SCREEN.H
　SCREEN.Cのプロトタイプ宣言
TEXT.H
　TEXT.Cのプロトタイプ宣言

今回も例によって，Borland C++ Ver2.0を使用している．リスト1にコンパイルのためのMAKEファイルを示す．このMAKEファイルを書き換えることによって，ほかのCコンパイラでもコンパイルが可能になると思う．BC++での実行ファイルの作成方法は，

```
A> MAKE -fBR.MAK
```

とする．これによって実行ファイル「BR.EXE」ができあがる．

使い方

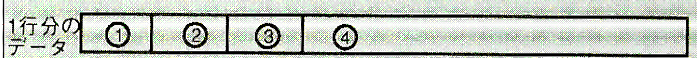
起動方法は以下のとおりである．

```
A> BR 表示ファイル名 [-tn]
```

起動オプション「t」は表示時のタブカラム数の指定である．nには，10進の数値が入る．このオプションを省略すると，4カラムごとのタブとなる．プログラムを起動すると，画面の2行目から12行目までの10行にテキストが表示される．テキストファイルが表示行数より大きい場合には，キーの無入力状態の時に順次読み込む．

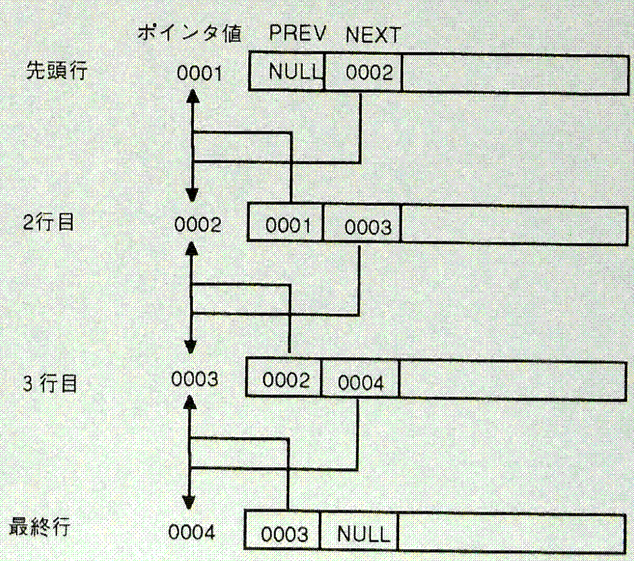
動作中に使用できるキー操作は以下

図2　行データの構造詳細



① … その行に含まれる文字の総バイト数
② … 前行の行データを指すポインタ（このデータが先頭行の場合はNULL）
③ … 次行の行データを指すポインタ（このデータが最終行の場合はNULL）
④ … 文字列本体

図3　それぞれの行のポインタでの連結状態



の3つである．

[CTRL]+ [E]
　1行スクロールダウンする
[CTRL]+ [X]
　1行スクロールアップする
[ESC]
　プログラムの終了

[CTRL]+[E]または[CTRL]+[X]を連続的に押すと，画面上部のみがスクロールしはじめる．キーから手を放した時点で10行すべてが書きかわる．これは，DOS汎用の標準的な画面出力手段を使用しているために，表示がまにあわなくなる状態を回避するためである．この部分のロジックは表示更新中にキー入力を監視するようになっているので，テキスト表示の非常に高速なマシンであれば，10行すべてがつねにスクロールするかもしれない．残念ながら筆者の使用しているマシンではキーを押しつづけると最上行が1行だけスクロールしている…….

今回は新テーマツールのご紹介ということで，文字列データ処理以外のプログラミング上の話題をいくつか取り上げることにする．プログラム起動時の動作を把握するためにメイン関数を解説したあと，キー無入力の間にファイルを読み込んでいく手法をとっているBrowsText関数，スクロールが連続している間は画面の書き換えを抑止するReWriteFullScreen関数，可変長タブ幅に対応したPutString関数，そして漢字コードが行末でちぎれないようにファイルリードしていくjfgets関数を取り上げる．これらの関数が，今後のプログラミングのヒントになれば幸いである．

メイン関数（リスト解説：図4）

オプションの解析，読み込むファイルのオープンを行い，ブラウザのメイン関数を実行する．

キー無入力時のファイルリード（リスト解説：図5）

この関数が，事実上のメイン関数である．ファイルを表示範囲に表示し，キーボード入力を監視する．有効なキー操作があれば，そのキーに対応した

リスト1　BR.MAK

```
.AUTODEPEND

#       *Translator Definitions*
CC = bcc +BR.CFG
TASM = TASM
TLINK = tlink


#       *Implicit Rules*
.c.obj:
  $(CC) -c {$< }

.cpp.obj:
  $(CC) -c {$< }

#       *List Macros*


EXE_dependencies =  ¥
  br.obj ¥
  brows.obj ¥
  screen.obj ¥
  text.obj

#       *Explicit Rules*
br.exe: br.cfg $(EXE_dependencies)
  $(TLINK) /x/c/P-/LA:¥BCPP¥LIB @&&|
c0s.obj+
br.obj+
brows.obj+
screen.obj+
text.obj
br
        # no map file
emu.lib+
maths.lib+
cs.lib
|


#       *Individual File Dependencies*
br.obj: br.c

brows.obj: brows.c

screen.obj: screen.c

text.obj: text.c

#       *Compiler Configuration File*
br.cfg: br.mak
  copy &&|
-v
-vi
-IA:¥BCPP¥INCLUDE
-LA:¥BCPP¥LIB
-P-.C
| br.cfg
```

リスト2　COMMON.H

```
/* ========================================================================
    COMMON.H            Copyright 1992 bY JUSTECH-LABO. (H.kanoshima)
                          (データ構造定義など全ソース共通定義)
======================================================================== */
typedef     enum {BFALSE,BTRUE} BOOL;

typedef struct  _Line{
    int     Length;
    struct  _Line *PrevLinePtr;
    struct  _Line *NextLinePtr;
    char    Body[1];
}   LINE;
```

リスト3　ROWS.H

```
/* ========================================================================
    BROWS.H         Copyright 1992 bY JUSTECH-LABO. (H.kanoshima)
                      (BROWS.C に含まれる関数のプロトタイプ宣言)
======================================================================== */
void    BrowsText(char *);
```

リスト4　SCREEN.H

```
/* ========================================================================
    SCREEN.H            Copyright 1992 bY JUSTECH-LABO. (H.kanoshima)
                         (SCREEN.C に含まれる関数のプロトタイプ宣言)
======================================================================== */
int     AskTabStep(void);
BOOL    ReWriteFullScreen(LINE *);
void    DisplayGuide(char *);
void    InitializeScreen(int, int, int);
```

リスト5　TEXT.H

```
/* ========================================================================
    TEXT.H          Copyright 1992 bY JUSTECH-LABO. (H.kanoshima)
                       (TEXT.C に含まれる関数のプロトタイプ宣言)
======================================================================== */
LINE    *AskNextLineAddress(LINE *);
LINE    *AskPrevLineAddress(LINE *);
LINE    *AskTopLineAddress(void);
LINE    *AskEndLineAddress(void);
char    *AskBodyAddress(LINE *);
void    ReadNextText(void);
BOOL    InitializeText(char *, int);
```


ワンラインNEWS
日本IBMの大手特約店であるトッパン・ムーアシステムズはIBM以外のコンピュータの販売を開始した．当初，サン・マイクロシステムズなどのWSなどを取り扱う．

処理を行う．また，キーが無入力であるときには，ファイルの未読部分の読み込みを行う．

連続スクロール処理（リスト解説：図6）

テキストの表示範囲に，引数で与えられたデータを表示する．ただし，表示中にキーボードが押下された場合には途中で処理を打ち切る．

タブの展開（リスト解説：図7）

テキストデータに含まれるタブコードをユーザーの指定するタブ幅に合わせて展開するロジックである．いうま

図4　メイン関数

```
int        main(int argc, char **argv)

（引数解説）
argc,argv .... 起動オプションのカウンタとその配列ポインタ。起動は「br ファイル名
[オプション･･･]」とし、ファイル名は必須なので、カウンタは最低でも2以上でなけれ
ばならない。

  : {
  :     if(argc < 2)
  :     {
  :         puts("Usage:br filename.ext [-tn]");
  :         return    ERROR_RETURN;
  :     }
 引数が2以上ないと、ファイル名が指定されていないのでエラー終了する。

  :     ProcessOption(argc, argv);
 起動オプションの解析。（このバージョンでは、「-t」の指定により、タブ幅が指定でき
る。

  :     InitializeScreen(LINETOP, LINEBOTOM, TabStep);
 表示桁数、行数、タブ幅などの表示範囲を決定する。

  :     if(InitializeText(argv[1], LINEBOTOM - LINETOP) == BFALSE)
  :         return    ERROR_RETURN;
 読み込むファイルをオープンし、最初の一画面分の読み込みを行なう。ファイルが見つ
からなければBFALSEを返してくる。

  :     BrowsText(argv[1]);
 ブラウザの本体関数を呼び出す。

  :     return    NORMAL_RETURN;
 正常終了。
  : }
```

リスト6　BR.C

```
/* ===================================================================
    実践C塾・テーマツール「BR.EXE」    メインソース

    BR.C            Copyright 1992 bY JUSTECH-LABO. (H.kanoshima)

=================================================================== */
#include    "common.h"
#include    "screen.h"
#include    "text.h"
#include    "brows.h"

#define     ERROR_RETURN        1
#define     NORMAL_RETURN       0

#define     LINETOP             2       /* 画面最上行 */
#define     LINEBOTOM           12      /* 画面最下行 */

#define     SWITCHCHAR          '-'

static      int TabStep = 4;

/* ===================================================================
 * 起動オプションを処理する。現在処理しているのは't'（TABステップの設定）
 * のみである。
 * ===================================================================
 */

static  void    ProcessOption(int argc, char **argv)
{
    argv++;
    while(--argc)
    {
    char    *Option;

        Option = *argv++;
        if(*Option++ == SWITCHCHAR)
        switch(*Option++)
        {
            case    't':
            case    'T':
                    TabStep = atoi(Option);
                    break;
            default:
                    break;
        }
    }
}

/* ============================
 * 言わずと知れたメインルーチン。
 * ============================
 */

int     main(int argc, char **argv)
{
    if(argc < 2)
    {
        puts("Usage:br filename.ext [-tn]");
        return  ERROR_RETURN;
    }

    ProcessOption(argc, argv);

    InitializeScreen(LINETOP, LINEBOTOM, TabStep);

    if(InitializeText(argv[1], LINEBOTOM - LINETOP) == BFALSE)
        return  ERROR_RETURN;

    BrowsText(argv[1]);

    return  NORMAL_RETURN;
}
```

リスト7　BROWS.C

```
/* ======================================================================
    実践C塾・テーマツール「BR.EXE」    ブラウザメイン処理ソース

    BROWS.C         Copyright 1992 bY JUSTECH-LABO. (H.kanoshima)

====================================================================== */
#include    <stdio.h>

#include    "common.h"
#include    "screen.h"
#include    "text.h"
#include    "brows.h"

#define     ESCAPEKEY   0x1b

#define     SCROLLDOWN  ('E' - '@')
#define     SCROLLUP    ('X' - '@')

/* ===================================================
 * ブラウザのメインルーチン。無入力時の残りテキストの読み込みは、
 * ここで処理している（ReadNextText()）
 * ===================================================
 */

void    BrowsText(char *FileName)
{
LINE    *DisplayLineAddress;
BOOL    ReWrite;

    DisplayGuide(FileName);
    DisplayLineAddress = AskTopLineAddress();
    ReWrite = ReWriteFullScreen(DisplayLineAddress);

    for(;;)
    {
        if(kbhit())
        {
        LINE    *lptr = NULL;
        int     Key;

            Key = getch();
            if(Key == ESCAPEKEY)    break;
            switch(Key)
            {
                case    SCROLLDOWN:
                        lptr = AskPrevLineAddress(DisplayLineAddress);
                        break;

                case    SCROLLUP:
                        lptr = AskNextLineAddress(DisplayLineAddress);
                        break;

                default:
                        break;
            }
            if(lptr != NULL)
            {
                DisplayLineAddress = lptr;
                ReWrite = ReWriteFullScreen(DisplayLineAddress);
            }
            else
            {
            if(ReWrite == BTRUE)
                ReWrite = ReWriteFullScreen(DisplayLineAddress);
            }
        }
        ReadNextText();
    }
}
```

でもなく，タブコードを固定でタブ幅分のスペースに置き換えるだけではいけない．現在の桁位置との関係から，次のタブ位置を計算するわけである．

漢字コードの扱い（リスト解説：図8）

テキストの閲覧ツールを作成する上で注意すべきなのは，漢字コードの取り扱いである．漢字コードは2バイトで1文字を表現するため，画面の端に漢字コードがきた場合に機種によっては表示が乱れることがある．これを回

リスト8 SCREEN.C

```
/* =====================================================================
    実践C塾・テーマツール「BR.EXE」   表示処理関連ソース

    SCREEN.C            Copyright 1992 bY JUSTECH-LABO. (H.kanoshima)

===================================================================== */
#include    <stdio.h>

#include    "common.h"
#include    "text.h"
#include    "screen.h"

#define     SCREENWIDTH     80

int     ScreenY0, ScreenY1, ScreenHight, ScreenWidth;

int     TabStep = 8;

/* =====================
 * ＴＡＢステップを返す。
 * =====================
 */
int     AskTabStep(void)
{
    return  TabStep;
}

/* ==========================================================
 * エスケープシーケンス出力用。先に0x1bを出力し、つぎに与えられた
 * 文字列を出力する。
 * ==========================================================
 */

static  void    Escape(const char *Sequence)
{
    putchar(0x1b);
    printf(Sequence);
}

/* ============================================
 * 指定行にカーソルを移動する。桁は１に固定される。
 * ============================================
 */

void    CsetY(int y)
{
char    Sequence[16];

    sprintf(Sequence, "[%d;%dH", y, 1);
    Escape(Sequence);
}

/* =======================================================
 * 表示範囲の最上行にカーソルを移動する。桁は１に固定される。
 * =======================================================
 */

void    CsetTopLine(void)
{
    CsetY(ScreenY0);
}

/* =======================================================
 * 表示範囲の最下行にカーソルを移動する。桁は１に固定される。
 * =======================================================
 */

void    CsetBotomLine(void)
{
    CsetY(ScreenY1);
}

/* ==========================================================
 * 表示範囲の最上行－１にカーソルを移動する。桁は１に固定される。
 * 上ガイドライン表示のため。
 * ==========================================================
 */

void    CsetTopGuideLine(void)
{
    CsetY(ScreenY0 - 1);
}

/* ==========================================================
 * 表示範囲の最下行＋１にカーソルを移動する。桁は１に固定される。
 * 下ガイドライン表示のため。
 * ==========================================================
 */

void    CsetBotomGuideLine(void)
{
    CsetY(ScreenY1 + 1);
}

/* ==================================
 * 現カーソル位置から右端までを消去する。
 * ==================================
 */

void    EreaseToLineEnd(void)
{
    Escape("[0K");
}

/* ==========================================================
 * 文字列を表示する。ＴＡＢステップが可変のため、ＴＡＢはそれに
 * 該当するだけの半角空白に置き換える。
 * ==========================================================
 */

void    PutString(char *StringPtr)
{
int     TabStep = AskTabStep();
int     Column = 0;

    while(*StringPtr != '¥0')
    {
        if(*StringPtr == '¥t')
        {
        int     i;

            Column += (i = TabStep - (Column % TabStep));
            for(; i; i--)
                putchar(' ');
        }
        else
        {
            putchar(*StringPtr);
            Column++;
        }
        StringPtr++;
    }
}

/* ==========================================================
 * 表示範囲にある全行を表示する。途中でキー入力があった場合は
 * そこで中断してBFALSEを返し、それ以外はBTRUE を返す。
 * ==========================================================
 */

BOOL    ReWriteFullScreen(LINE *LinePtr)
{
int     LineCount;

    CsetTopLine();
    LineCount = ScreenHight;

    while(LineCount--)
    {
    char    *StringPtr;

        if(LinePtr != NULL)
        {
            PutString(AskBodyAddress(LinePtr));
            LinePtr = AskNextLineAddress(LinePtr);
        }
        EreaseToLineEnd();
        putchar('¥n');

        if(kbhit()) return  BTRUE;
    }
    return  BFALSE;
}

/* ==================================
 * 表示範囲上下のガイドラインを表示する。
 * ==================================
 */

void    DisplayGuide(char *FileName)
{
int Count;

    CsetTopGuideLine();
    printf("[ %s ] ", FileName);
    Count = ScreenWidth - strlen(FileName) - 5;
    while(Count--)
        putchar('-');
    CsetBotomGuideLine();
    Count = ScreenWidth;
    while(Count--)
        putchar('-');
}

/* ==============================
 * 本モジュールの初期値を設定する。
 * ==============================
 */

void    InitializeScreen(int y0, int y1, int tabstep)
{
    TabStep = tabstep;

    ScreenY0 = y0 + 1;
    ScreenY1 = y1 - 1;

    ScreenWidth = SCREENWIDTH;
    ScreenHight = ScreenY1 - ScreenY0 + 1;
}
```


ワンラインNEWS
オムロンは日本電気のパソコン「98note」専用の内蔵型モデム「MD24FN5V」を発売した．従来の製品にデータ圧縮機能の標準であるV.42bisを搭載した．

リスト9　TEXT.C

```c
/* =====================================================================
    実践C塾・テーマツール「BR.EXE」　テキスト処理関連ソース

    TEXT.C          Copyright 1992 bY JUSTECH-LABO. (H.kanoshima)

===================================================================== */
#include    <stdio.h>

#include    "common.h"
#include    "screen.h"
#include    "text.h"

#define     LINEMAXLENGTH       80
#define     LINEMAXLENGTHZ      (LINEMAXLENGTH + 1)
#define     LINESIZE            (sizeof(LINE))
#define     PATHLENGTH          80
#define     NEWLINEFLAG         0x00
#define     CONTLINEFLAG        0x01

LINE    *TopLine, *EndLinePtr, *CurLinePtr;
char    EditFileName[PATHLENGTH];

FILE    *FilePtr;
BOOL    ExistNextText;

/* =======================================
 * ヘッダも含めた一行分のメモリを獲得する。
 * =======================================
*/

static  LINE    *AllocateLine(int BodySize)
{
    return  (LINE *)malloc(LINESIZE + BodySize);
}

/* ==================================
 * カレント行ポインタをひとつ進める。
 * ==================================
*/

static  void    MoveCurLinePtrNext(void)
{
    if(CurLinePtr->NextLinePtr == NULL) return;
    CurLinePtr = CurLinePtr->NextLinePtr;
}

/* ==========================
 * つぎの行のポインタを得る。
 * ==========================
*/

LINE    *AskNextLineAddress(LINE *LinePtr)
{
    return  LinePtr->NextLinePtr;
}

/* ========================
 * 前の行のポインタを得る。
 * ========================
*/

LINE    *AskPrevLineAddress(LINE *LinePtr)
{
    return  LinePtr->PrevLinePtr;
}

/* ==========================
 * 先頭の行のポインタを得る。
 * ==========================
*/

LINE    *AskTopLineAddress(void)
{
    return  TopLine;
}

/* ==========================
 * 最後の行のポインタを得る。
 * ==========================
*/

LINE    *AskEndLineAddress(void)
{
    return  EndLinePtr;
}

/* ==========================================
 * 行ポインタから文字列の先頭アドレスを得る。
 * ==========================================
*/

char    *AskBodyAddress(LINE *LinePtr)
{
    return  LinePtr->Body;
}

/* ==============================
 * 指定された文字列を一行追加する。
 * ==============================
*/

static  BOOL    AddLine(char *StringPtr)
{
LINE    *LinePtr;

    LinePtr = AllocateLine(strlen(StringPtr));
    if(LinePtr == NULL)     return  BFALSE;

    strcpy(LinePtr->Body, StringPtr);

    if(TopLine == NULL)
    {
        TopLine = LinePtr;
        LinePtr->NextLinePtr = NULL;
        LinePtr->PrevLinePtr = NULL;
        CurLinePtr = TopLine;
    }
    else
    {
        LinePtr->NextLinePtr = NULL;
        LinePtr->PrevLinePtr = EndLinePtr;
        EndLinePtr->NextLinePtr = LinePtr;
    }
    EndLinePtr = LinePtr;

    return  BTRUE;
}

/* ====================================================
 * バイト数指定、ただし漢字が1バイト目でちぎれないように
 * ファイルから一行読み込む。TABはそれに該当する分だけ
 * 空白を読み込んだことにしてバイト数を計算する。
 * ====================================================
*/

static  char    *jfgets(char *BufferPtr, int Length, FILE *FilePtr)
{
int     TabStep = AskTabStep();
int     ByteCount = 0;
char    *BufferTop = BufferPtr;
BOOL    KanjiFlag = BFALSE;

    Length--;

    while(ByteCount < Length)
    {
    int     c;

        c = fgetc(FilePtr);
        switch(c)
        {
            case    EOF:
                        if(ByteCount == 0)
                            return  NULL;
                        else
                        {
                            *BufferPtr = '¥0';
                            return  BufferTop;
                        }
            case    '¥n':
                        *BufferPtr = '¥0';
                        return  BufferTop;
            case    '¥t':
                        *BufferPtr++ = c;
                        ByteCount += TabStep - (ByteCount % TabStep);
                        KanjiFlag = BFALSE;
                        break;
            default     :
                        if(!iskanji(c) || KanjiFlag == BTRUE)
                        {
                            *BufferPtr++ = c;
                            ByteCount++;
                            KanjiFlag = BFALSE;
                        }
                        else
                        {
                            if(Length - ByteCount < 2)
                            {
                                ungetc(c, FilePtr);
                                *BufferPtr = '¥0';
                                return  BufferTop;
                            }
                            else
                            {
                                *BufferPtr++ = c;
                                ByteCount++;
                                KanjiFlag = BTRUE;
                            }
                        }
        }
    }
    *BufferPtr = '¥0';
    return  BufferTop;
}

/* ==========================================
 * ファイルから一行読み込んでリストに追加する。
 * ==========================================
*/

void    ReadNextText(void)
{
char    ReadBuffer[LINEMAXLENGTHZ];

    if(jfgets(ReadBuffer, LINEMAXLENGTHZ, FilePtr) == NULL)
    {
        fclose(FilePtr);
        ExistNextText = BFALSE;
    }
    else
    {
        if(AddLine(ReadBuffer) == BFALSE)
        {
            ExistNextText = BFALSE;
            fclose(FilePtr);
        }
        else
            MoveCurLinePtrNext();
    }
}

/* ==================================================================
 * ファイルから表示範囲に相当する行数だけ読み込んでリストに追加する。
 * ==================================================================
*/

```

```
static  void    ReadInitialText(int InitialReadCount)
{
char    ReadBuffer[LINEMAXLENGTHZ];

    ExistNextText = BTRUE;

    while(jfgets(ReadBuffer, LINEMAXLENGTHZ, FilePtr) != NULL)
    {
        if(!InitialReadCount--)
            break;
        if(AddLine(ReadBuffer) == BFALSE)
        {
            ExistNextText = BFALSE;
            fclose(FilePtr);
            break;
        }
        MoveCurLinePtrNext();
    }
}

/* ======================================================================
 * このモジュールを初期化する。主な処理は変数の初期化、ファイルのオープンと
 * 表示範囲分の行数の読み込み。
 * ======================================================================
*/

BOOL    InitializeText(char *FileName, int InitialReadCount)
{
    strcpy(EditFileName, FileName);
    FilePtr = fopen(EditFileName, "rt");    /* テキストモードリード */
    if(FilePtr == NULL)
    {
        printf("ファイルがありません(%s)¥n", FileName);
        return  BFALSE;
    }
    EndLinePtr = CurLinePtr = TopLine = NULL;

    ReadInitialText(InitialReadCount);

    return  BTRUE;
}
```

図5　ブラウザのメイン関数

```
void    BrowsText(char *FileName)

（引数解説）
FileName ... 読み込むファイル名文字列へのポインタ

  : {
  : LINE    *DisplayLineAddress;
  表示する行データへのポインタ

  : BOOL    ReWrite;
  キーが押されて表示が途中になっているときにBTRUEになる。

  :     DisplayGuide(FileName);
  上下のガイドラインを表示する。上のガイドラインにはファイル名が入るため、ファイ
  ル名文字列へのポインタを渡す。

  :     DisplayLineAddress = AskTopLineAddress();
  最初の表示は、最上行から始まる。

  :     ReWrite = ReWriteFullScreen(DisplayLineAddress);
  最初の1画面を表示する。途中でキーが押されたときには表示が中断し、ReWriteがBTR
  UEになる。

  :     for(;;)
  :     {
  ここから終了キー操作が行われるまでくり返し処理となる。

  :         if(kbhit())
  :         {
  :         LINE    *lptr = NULL;
  :         int         Key;
  :
  以降は、キーボードが押された場合の処理

  :             Key = getch();
  押されたキーデータを読み取る。

  :             if(Key == ESCAPEKEY)    break;
  エスケープキーが押されたらこのくり返し処理をぬけ、プログラムは終了する。

  :             switch(Key)
  :             {
  :                 case    SCROLLDOWN:
  :                         lptr = AskPrevLineAddress(DisplayLineAddress);
  :                         break;
  スクロールダウンキー操作の場合には、一つ前の行から表示を更新する。

  :                 case    SCROLLUP:
  :                         lptr = AskNextLineAddress(DisplayLineAddress);
  :                         break;
  スクロールアップキー操作の場合には、一つ後の行から表示を更新する。

  :                 default:
  :                         break;
  その他のキーの場合は、無視する。

  :             }
  :             if(lptr != NULL)
  :             {
  :                 DisplayLineAddress = lptr;
  :                 ReWrite = ReWriteFullScreen(DisplayLineAddress);
  表示すべき行データであれば、表示を行なう。

  :             }
  :             else
  :             {
  :             if(ReWrite == BTRUE)
  :                     ReWrite = ReWriteFullScreen(DisplayLineAddress);
  以前に中断していた表示があるなら、再度表示を行なう。

  :             }
  :         }
  以降は、キーボードが押されなかった場合の処理。
  :         ReadNextText();
  ファイルの未読部分を読み込む。

  :     }
  : }
```

図6　連続スクロール処理

```
BOOL    ReWriteFullScreen(LINE *LinePtr)

（引数解説）
LinePtr ... 表示を開始する行データへのポインタ

（戻り値解説）

BTRUE ... 途中でキーが押されて中断した。
BFALSE... 表示範囲内をすべて表示した。

  : {
  : int         LineCount;
  表示すべき行数を格納するエリア

  :     CsetTopLine();
  カーソルを表示範囲の最上行・1桁目に移動する。

  :     LineCount = ScreenHight;
  表示すべき行は、表示範囲行数に等しい。
  :
  :     while(LineCount--)
  :     {
  :     char    *StringPtr;
  表示すべき行数だけ処理をくり返す。

  :         if(LinePtr != NULL)
  :         {
  データが存在するときだけ表示を行なう。

  :             PutString(AskBodyAddress(LinePtr));
  一行分の表示を実行する。

  :             LinePtr = AskNextLineAddress(LinePtr);
  :         }
  データのポインタをつぎの行へ移動させる。

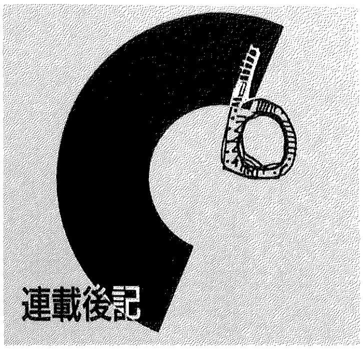
  :         EreaseToLineEnd();
  上書きしているので、最後に表示した文字から行末までをクリアする。

  :         putchar('¥n');
  改行する。

  :         if(kbhit())    return    BTRUE;
  一行表示するごとにキーボードが押されているかどうかをチェックし、押されていたら
  そこで処理を中断する。

  :     }
  :     return    BFALSE;
  表示範囲をすべて表示したら終了する。
  : }
```

避するために，本プログラムでは画面の右端に漢字コードの1バイト目がきたときには，このデータを次の行の先頭にずらす処理を行っている．



連載後記

アルゴリズムは，決して机上の学問だけではなく，プログラミングの実戦においても重要なものであることを，これまでの連載を通じてお伝えしてきた．そして今回から2回にわたって，このことを実際のプログラムを使って体感していただくために，応用編ともいうべき可変長の文字列データを扱うツールがどのようなアルゴリズムを消化しているかを見ていくことになる．来月は，テーマツールで使用しているデータ構造をもとに，「リスト構造」の手法をさらに掘り下げて考察していくことにしよう．

図7　文字列バッファから画面表示を行う

```
void    PutString(char *StringPtr)

 (引数解説)
StringPtr ... 文字列の格納ポインタ

  : {
  : int         TabStep = AskTabStep();
  タブ幅は、起動オプションによって可変になっているので初期値をここで設定する

  : int         Column = 0;
  タブ位置を算出するために、現在の桁位置を記録するワーク

  :     while(*StringPtr != '¥0')
  :     {
  文字列の終端にはNULLが格納されている。

  :         if(*StringPtr == '¥t')
  :         {
  :         int         i;
  :
  :             Column += (i = TabStep - (Column % TabStep));
  :             for(; i; i--)
  :                 putchar(' ');
  タブコードの場合は、現在の桁位置から次のタブ位置までの間に半角スペースをいくつ
必要かを計算して表示する。

  :         }
  :         else
  :         {
  :             putchar(*StringPtr);
  :             Column++;
  タブ以外の文字はそのまま表示し、桁位置をひとつ進める。

  :         }
  :         StringPtr++;
  文字列のポインタを一つすすめる。

  :     }
  : }
```

図8　指定されたバイト数を上限としてファイルからテキストをリードする

```
static    char     *jfgets(char *BufferPtr, int Length, FILE *FilePtr)

 (引数解説)
BufferPtr ... 読み込んだ文字列の格納ポインタ
Length    ... 上記格納領域のバイト数
FilePtr   ... ファイルの読み込みポインタ

 (戻り値解説)
・データを正常に読み込んだ場合には、読み込んだ文字列の格納ポインタを返す
・データを読み込めなかった場合にはNULLを返す

  : {
  : int         TabStep = AskTabStep();
  タブ幅は、起動オプションによって可変になっているので初期値をここで設定する

  : int         ByteCount = 0;
  実際に読んだバイト数を積算するエリア

  : char      *BufferTop = BufferPtr;
  読み込み領域の先頭を記憶するエリア

  : BOOL      KanjiFlag = BFALSE;
  漢字（2バイトコード）を処理するためのフラグ

  :     Length--;
  最後にNULLを付加するため

  :     while(ByteCount < Length)
  :     {
  :     int         c;
  :
  :         c = fgetc(FilePtr);
  ファイルから1バイト読み込む

  :         switch(c)
  :         {
  :             case    EOF:
  :                         if(ByteCount == 0)
  :                             return    NULL;
  :                         else
  :                         {
  :                             *BufferPtr = '¥0';
  :                             return    BufferTop;
  :                         }
  文末コードをリードした場合には、1バイトも読んでいない場合にはNULLを返して
終了する。データが1バイト以上あった場合には、文字列の末尾にNULLを付加して、
データ領域の先頭ポインタを返す。

  :             case    '¥n':
  :                         *BufferPtr = '¥0';
  :                         return    BufferTop;
  改行コードをリードした場合には、それ自身がデータとなるので、文字列の末尾にNU
LLを付加して、データ領域の先頭ポインタを返す。

  :             case    '¥t':
  :                         *BufferPtr++ = c;
  :                         ByteCount += TabStep - (ByteCount % TabStep);
  :                         KanjiFlag = BFALSE;
  :                         break;
  タブコードをリードした場合には、表示時のタブ幅から画面上でいくつ空白が必要かを
計算し、該当するバイト数だけ読み込んだことにする。ここでは実際に空白を入れること
はしていない。これはデータ格納領域のメモリ消費を節約するためである。

  :             default     :
  :                         if(!iskanji(c) || KanjiFlag == BTRUE)
  :                         {
  :                             *BufferPtr++ = c;
  :                             ByteCount++;
  :                             KanjiFlag = BFALSE;
  :                         }
  漢字以外のデータおよび漢字コードの2バイト目は無条件に1バイトデータとして文字
列に追加する。漢字フラグはリセットされる。

  :                         else
  :                         {
  :                             if(Length - ByteCount < 2)
  :                             {
  :                                 ungetc(c, FilePtr);
  :                                 *BufferPtr = '¥0';
  :                                 return    BufferTop;
  :                             }
  漢字コードの1バイト目の場合には、このデータを含んであと2バイトのデータが格納
できるかどうかチェックする。1バイトしか空きがない場合（画面上で右端のデータとな
る）には、このデータをファイルに戻し、処理を終了する。

  :                             else
  :                             {
  :                                 *BufferPtr++ = c;
  :                                 ByteCount++;
  :                                 KanjiFlag = BTRUE;
  :                             }
  2バイト以上空きがある場合には、このデータを通常と同じように文字列に追加する。
ただし、後続の1バイトは無条件に処理できるように漢字フラグをたてる。

  :                         }
  :         }
  :     }
  :     *BufferPtr = '¥0';
  :     return    BufferTop;
  改行または文末コードを読み込む前に、バイトカウンタがリミットに達したら、それま
でに読み込んだ文字列の末尾にNULLを付加して、処理を中断する。

  : }
```

98活用アセンブラ技法

キーボード編
第1回

キーボードロック／リセットプログラムの作成

星野 操

画面・タイマ・SCSI・シリアルと，PC-9801のBIOSの拡張やそれを利用したプログラムを記述してきたが，連載の最後に，キーボード関係のBIOSについて解説・拡張をほどこしてみる．今回は，キーコンビネーションによるリセット機能とキーロック機能を実現する常駐プログラムを作成する．

最新アセンブラ事情

まず最初に，TASM 3.0が入手できたので，これについて少し触れることにします．TASM 3.0は現在はBorland C++ 3.0のパッケージに入っています．現時点ではBC++ 3.0はIBM版しかリリースされていませんが，もうすぐ98版もリリースされることになっています．

ちなみに，IBM版のTASM 3.0は98では動作しないようです．2.0以前でも内部的なパッチエントリベクタとキーボードBIOSベクタがコンフリクトするために，問題が生じることはありました．しかし3.0ではよりIBM PCに依存しているようで，変更なしでは98上ではメッセージも出さずにハングアップしました．

TASM 3.0とほぼ同時期に，MASMも6.0にバージョンアップされました．このMASM 6.0については，5月号に詳しい解説が載りました．その改良点を大雑把にまとめると，

- **Cなどとの相性のより一層の改善：構造体などの機能変更・拡張・変換プログラム**
- **条件判断のための疑似命令の強化**

ということになるでしょう．とくに後者の条件判断疑似命令の強化は目を見張るものがあります．MASM 6.0でサポートされた疑似命令は，同様の目的と制御構造を持ったものがすでにマクロで実現されてはいます．しかし大きく異なるのは，条件式の記述能力でしょう．マクロで実現されたものの内，おもなものは条件を生成する1つ（またはそれ以上）のアセンブラコマンドと，判断する条件（ZやCなど）とを併記するものでした．これに対して，MASM 6.0では==などのCライクな演算子を直接条件に記述できるようになっています．もちろん，このような機能をサポートしているマクロもないわけではありませんでしたが少数派といえるでしょう．ましてや，MASM 6.0がサポートしている&&や||などの複合条件まで満たしているものはほとんどなかったといえます．

さらに，これらの機能はシステムサポートされているため，演算の最適化も少なからず行われています．生成されるコードの予測はできませんが，比較的質の高いオブジェクトを容易に取得できるメリットは，大きいといえるでしょう．

オブジェクト指向アセンブラの機能

これに対して，TASM 3.0の改良はかなり地味であるといえます．TASM 3.0の最大の特長は，OOA (Object Oriented Assembler) ということになります．しかし，このOOAは別にC++とインターフェースを取るためのものではありません．

異なるクラスに同一名称のメソッドを定義することは一般的に行われます．この場合，それぞれを区別して呼び出す必要があります．このため，C++ではメソッド（関数）の名前を細工して，コンフリクトが生じないようにしてあります．さらに場合によっては引数の型を名前の修飾子に付けて，複数のモジュール間で引数のタイプの不整合が起きないようにしてあります．

つまり，クラスfooのメソッドbarの外部参照名は，Cのように_barではなくもっと複雑な名前になっています．この変換形式を知っていなければ，リンク可能な名前を付けることはできませんし，デバッガなどでも指定のシンボルを参照することができません．

TASM 3.0がOOP対応になることによって，このような「名前の変更」の方法をサポートし，アセンブラからC++のメソッドが，またC++からアセンブラのメソッドが簡単に呼び出せれば，非常にありがたかったといえます．しかし現実はそのようにはなっていません．マニュアルでは，C++がアセンブラとインターフェースを取る場合には，呼び出したり呼び出されたりする関数名を“C”形式として宣言し，名前の特別な変更が一切なされないようにしないといけないとあります．この場合，同一のメソッドを複数のクラスで定義することはできないので，ユニークな名前を付けたアセンブラやC形式の「関数」をこれらクラスのメソッドで呼び出す，というふうにしないといけません．

ではTASM 3.0がどのようなOOPに対応しているかというと，これは完全にアセンブラの世界で閉じているものです．クラスはアセンブラではSTRUC構造体を拡張した形で定義します．これを定義する時に，以前のようにメンバ変数のほかに，メソッドテーブルを記述します．ここで注意するのは，定義するのは，あくまでテーブルであってメソッド本体ではないということです．メソッド本体は別にユニークな名前を付けたPROCとして定義しなければいけません．そして，そのアドレスをテーブルにして，ユニークとは限らないメソッド名として参照するわけです．

メソッドを呼び出す時には，CALL命令を使用します．ただし，呼び出し形式は普通のCALL命令とはかなり

異なります．CALL命令は，呼び出し先を第一引数として取ります．しかしメソッドの呼び出しでは，まず第一引数にオブジェクトへのポインタなど，次にキーワードMETHODとそれに続いてメソッド名を記述します．それから引数のリストを記述します．

オブジェクトへのポインタは，一般的にレジスタインダイレクトで示されます．この表記では，指し示しているオブジェクトのクラスがわかりません．ですからメソッド名はクラス名を合わせて明示する必要が出てきます．メソッドは，処理するオブジェクトと対応付けられていますから，通常はそのオブジェクトのアドレスをメソッドの引数として渡すことになります．

メソッドには，静的にアドレスが決定されるものと，動的に決定されるものとがあります．動的に決定すべきメソッドは，リスト宣言の時VIRTUALキーワードを付加します．これはC++と同様です．メソッドの属するクラス名が明示されていることから，静的メソッドの場合はテーブルを参照することなしに直接CALL命令が作成されます．逆にいえば，オブジェクトを指すCALL命令の第一引数は事実上使用されません．

これに対して，動的メソッドは，オブジェクトのメソッドテーブルを参照し，そこから呼び出しアドレスを計算します．このためいくつかのレジスタが破壊されます．この時に使用するレジスタはCALL命令の引数で指定できます．このような方法を使用すれば，同様のメソッドリスト構造を持つ異なるクラスのポインタから，対応するメソッドを呼び出すことができます．

list.asm

参考までにこのOOPの機能を利用したアセンブラのリストを示します（リスト1）．リストを構成するためのセルの定義と，それを利用した簡単なメソッドです．最初にクラスcellの定義があります．クラスの定義はSTRUC疑似命令を拡張したもので，最初にキーワードMETHODとそれに続くメソッドテーブルの定義があります．テーブルは {} で括られ，メソッド名とそれに対応する実際のPROCエントリが記述されます．ここでword指定はメソッドがnearコールされるということを示します．dword指定がされていると，farコールされます．ここでは，メソッドcarとcdrはVIRTUAL指定してあります．これらはテーブル参照で間接的に呼び出されますが，その他のメソッドは対応するPROCが直接呼び出されます．最後にメンバ変数を2つ定義します．そのすぐ後のTBLINSTは，仮想関数呼び出しテーブルを実際にインプリメントするためのキーワードです．このテーブルのアドレスは，各オブジェクトインスタンスの見えないメンバ関数の中にセットされます．

次にセルヒープと，フリーリストのポインタ変数を用意します．clear_heap_initのPROCは，フリーリストを作成する手続きで，最初に1度呼び出します．これでヒープアレイをリンクしてフリーリストとします．その次のcell_newは，メソッドnewに対応付けられた手続きです．メソッドnewが呼び出されると，実際にはこのcell_newが呼び出されることになります．この手続きは，デフォルトオブジェクトの引数を取らないようにします．手続きは，フリーリストからセルを1つ取り出して，そのアドレスをAXに入れて返します．

cell_newの中では，メソッドcdrを使用しています．cdrは動的メソッドなので，オブジェクトのテーブルを参照して呼び出されます．このため，オブジェクトのアドレスを明示しないといけません．ここではBXレジスタにフリーセルの先頭アドレスを設定し，それを経由します．このアドレスを示すBXレジスタの後にキーワードMETHODを記述します．その後に，cellというクラス名指定を付加した上でcdrというメソッド名を記述します．cell_cdrを直接呼び出すことはしません．なお，ここで仮想関数を呼び出すためにデフォルトでES:BXレジスタペアが使用（破壊）されるので，ここではUSESキーワードを用いて，DS:SIを使用するようにしてあります．最低でもDSレジスタを示さないと，DS==ESでない場合には，正常に動作しなくなります．なお，ほかのCALL命令は静的に確定したメソッドなのでテーブル参照は行われません．ですから，USES指定は省いてしまっています．

このメソッドcdrはオブジェクトアドレスを引数として取るので，BXレジスタを引数として指定します．このPROCの拡張はTASM 2.0からのもので，PROCの呼び出しアドレスの後に引数リストを列挙することで，自動的にスタックにプッシュを実行します．呼び出し形式によって，Cのように左から右，Pascalのように右から左のプッシュの双方に対応し，さらにC形式の場合は自動的に引数のポップアウトコードまで生成されます．なお，その後ろに出てくるTBLINITは，見えないメンバ変数にテーブルのアドレス

をセットする一種のマクロです．仮想関数を使用する場合には，必ずこのキーワードを使用する必要があります．

cell_car, cell_cdrはそれぞれcar, cdrに対応するPROCで，テーブル参照で呼び出されます．これらはオブジェクトを指すデフォルトのポインタ引数を取ります．そして，SIレジスタにまずそのアドレスを取り，そのオブジェクトのメンバ変数をAXに読み込んで返します．

cell_set_car, cell_set_cdrは値をセットする関数で，デフォルトポインタのほかにセットする値を取ります．これもSIにアドレスをセットしてから，そのメンバ変数に値をセットします．

cell_constructはコンストラクタですが，別にC++のように自動的に呼び出されるというわけではありません．デストラクタでも同様です．コンストラクタはまず，メソッドnewを呼び出します．ここでは第一引数にcell_heapを取っていますが，これは実際にはまったくのダミーです．メソッドcarのようにテーブル経由以外のメソッドの場合は，PROCの第一引数は実質的に意味をもちません．その後carとcdrに値を設定します．これらはコンストラクタの第一引数と第二引数として渡されているものです．set_carとset_cdrのためのCALL命令の第一引数に取るオブジェクトアドレスとして，SIレジスタにnewの返り値を設定していますが，実際には使用されません．ただし，set_car/set_cdrへの引数としてオブジェクトのアドレスを渡す必要はあります．

cell_destroyはデストラクタで，渡されたオブジェクトをフリーリストにリンクします．実際にプログラムを作成する場合には，これらの基本的なメソッドに加えて，より高度なメソッドをクラスの中に定義したり，その外側でメソッドを利用してプログラムを記述したりすることになります．

というわけで，大雑把にOOAの機能を見てみましたが，どうもあまりありがたい機能とはいいがたいものがあります．というのも，メソッドには必ずユニークな名前を持つ手続きを対応付ける必要がありますし，C++でいう

リスト1　list.asm

```

        .model  small,pascal

        .data

cell STRUC GLOBAL METHOD {
        construct:word = cell_construct
        destroy:word = cell_destroy
        new:word = cell_new
        set_car:word = cell_set_car
        set_cdr:word = cell_set_cdr
        virtual car:word = cell_car
        virtual cdr:word = cell_cdr
        }

        car_ptr dw      ?
        cdr_ptr dw      ?
ENDS
        TBLINST

CELL_COUNT      =       1024

cell_heap       cell    CELL_COUNT dup ()
cell_heap_tail  label   cell

free_cell_ptr   dw      0

        .code

cell_heap_init  PROC
USES    bx,dx
        mov     free_cell_ptr,offset cell_heap

        mov     dx,offset cell_heap
        l1:
                mov     bx,dx
                add     dx,size cell
                mov     [bx].cdr_ptr,dx
                cmp     dx,offset cell_heap_tail
        jb      l1
        mov     [bx].cdr_ptr,0

        ret
cell_heap_init  ENDP

cell_new        PROC
USES    bx, si
        cmp     free_cell_ptr,0
        jnz     l2
                call    cell_heap_init
        l2:
        mov     bx,free_cell_ptr
        call    bx METHOD cell:cdr USES ds:si, bx
        or      ax,ax
        jz      l3
                ; allocatable
                mov     free_cell_ptr,ax
                TBLINIT bx
                mov     ax,bx
        l3:
        ret
cell_new        endp

cell_car        PROC
ARG     @@cell
USES    si
        mov     si,@@cell
        mov     ax,[si].car_ptr
        ret
cell_car        ENDP

cell_cdr        PROC
ARG     @@cell
USES    si
        mov     si,@@cell
        mov     ax,[si].cdr_ptr
        ret
cell_cdr        ENDP

cell_set_car    PROC
ARG     @@cell,value
USES    si
        mov     si,@@cell
        mov     ax,value
        mov     [si].car_ptr,ax
        mov     ax,si
        ret
cell_set_car    ENDP

cell_set_cdr    PROC
ARG     @@cell,value
USES    si
        mov     si,@@cell
        mov     ax,value
        mov     [si].cdr_ptr,ax
        mov     ax,si
        ret
cell_set_cdr    ENDP

cell_construct  PROC
ARG     car_value,cdr_value
USES    si
        call    cell_heap METHOD cell:new
        mov     si,ax
        call    si METHOD cell:set_car, si car_value
        call    si METHOD cell:set_cdr, si cdr_value
        mov     ax,si
        ret
cell_construct  ENDP

cell_destroy    PROC
ARG     @@cell
USES    si
        mov     si,@@cell
        call    si METHOD cell:set_cdr, si free_cell_ptr
        mov     free_cell_ptr,si
        ret
cell_destroy    ENDP

        end
```

ようなインラインメソッドの機能がありませんので，カプセル化を推し進めるとメンバへの簡単なアクセスでも，かなりオーバーヘッドの大きい関数呼び出しとして実現する必要が出てきます．前出のような例の場合，アセンブラで記述するよりはC++で記述したほうがタイトなコードを生成することができます．どうも，あまりアセンブラで高度な記述をしようというのは，向いていないような気がします．それならば，C++などとのインターフェースについて配慮するほうが先でしょう．今回のバージョンアップに関していえば，MASM 6.0のほうに軍配が上がるような気がします．

PC-9801のキーボードBIOS

さて，画面・タイマ・SCSI・シリアルと，そのシステムのチェックや解説，そしてBIOSの拡張やそれを利用したプログラムを記述してきました．最後に，キーボード関係のBIOSについて解説・拡張をほどこしてみようと思います．PC-9801の場合，キーボードは8048などのCPUによって制御され，本体とは19200bpsのシリアルラインで結合されています．

キーボードが押されたり離されたりすると，個々のキーに対応したコードが本体に送出されます．送出されるコードの下位7ビットはキーに対応しており，最上位ビットが1ならキーが離された（ブレーク）ことを，0ならキーが押された（メーク）ことを示します．キーのリピート処理はキーボードの内部で行われており，実際にキーがリピートしている状態では，キーのメークとブレークのコードを繰り返し間欠的に送出しています．この処理は内部で完結しているので，基本的にリピート間隔やリピートするキーを変えることはできません．

本体のソフトウェアでそれなりに対応することは不可能ではありません．特定の間隔でキーのメーク・ブレークが繰り返された場合，それはリピートしていると考えてリピート処理を止めたり，あるいは一定時間キーが押されたままになっているとキーが押されたのと同様な処理を実行したりするわけです．しかし，よほどうまく作らないとよく動作しません．

以前あるエディタで，カーソル移動をしている時に，リピートが始まるとキーのリピート速度より高速でカーソルが動くというものがありました．これは前記のようにリピート間隔を見て，それによってキーを「水増し」していたのですが，認識が甘くてカーソルを2度叩くと，そのタイミングにより4つ程度もカーソルが動いてしまうということがありました．

シフトキーの処理，またSTOP/COPYなどの特殊キーの処理は，すべてBIOSの中で処理されています．シフトキーは，キーボードから見るとまったく通常のキーと同一の処理がなされているわけです．BIOSでシフト状態を判断することにより，それに対応したASCIIコードを作成します．

Rxシリーズ以降のキーボードは，ソフトウェアでCAPS/カナなどのシフト状態が変更できるようになりました．これはタスクスイッチをした場合など，タスクによってシフト状態を一定にできるようにするためのもののようです．このため，これらのキーは物理的なトグルキーではなく，単なるキーになり，シフト状態はLEDで表示されるようになりました．また，シフト状態はメモリスイッチと同様に不揮発メモリに格納されるため，リセット・パワーオン時などにも前の状態の復元がなされます．

確かに，カルクとワープロを切り替えて起動し，ワープロではカナキーOFF，カルクではカナキーONになっている状態で使いたい，という場合には意味があるかもしれません．しかし，タスクスイッチによって自動的にキーシフト状態が変わるというのも逆に不便なような気がします．

いずれにしろ，これを可能ならしめるために，キーボードにコマンドを送出し，キー状態を変更できるようにキーボード内のプログラムの変更がなされました．ただし，これらは「可能である」ということであって，「必要である」ということではありません．つまり，シフト状態・LEDの処理は基本的にはキーボードの内部で処理されており，それを強制的に変更する場合にのみコマンドを送ればよいわけです．ですから，普通にキーボードを処理する分にはあえて使用する必要はないものです．

キーボードとのデータのやり取りは，ハンドシェークによって行われます．キーボードは，インターフェース信号線/RDYが0になっている時にのみキースキャンデータを本体に送ることができます．この信号線/RDYは，シリアルICのRxRDYとRTSがともに0である時にのみ0になります．ですから，RTSをONにすることによって，キーボードの処理をペンディングすることができます．

キーボードからデータが送られると，シリアルICのRxRDYが自動的にセットされます．これによって/RDYが1になるわけですが，キーボードはデータ転送後にこの信号線をワッチし，それが0になるまで待ちます．待

っている間は，LEDの制御などは行いません．たとえばカナキーを押した場合，まずそのスキャンコードデータを本体に送出し，本体がそれを受け取った後にLEDの点滅が行われます．ですから本体が割り込み禁止になっている場合など，LEDの点滅操作は行われません．

本体が送出したデータを受け取ったことは，前記のように/RDYによって把握されます．この時，/RTY(シリアルICのDTR）がリセットされていれば，そのまま次の処理に入ります．しかし，これがセットされていれば，同じキーコードを再送します．万一キーボードからのシリアルデータがパリティエラーなどエラー状態になった場合，この機能を用いて再送要求をするわけです．ですから，最初にシリアルICのステータスをチェックし，それによって/RTYの値を設定してから，次に実際に送られたデータを受け取るという手順を取らないといけません．なお，この/RDYの「下がりエッジ」を捕らえる十分な時間をキーボードに与えるために，データが受信されてから受け取るまでの間に十分な時間をおかないといけません．

キーボードのトラップ

キーボードをトラップするということは，比較的よくなされます．たとえば，特定のキーコンビネーションが押された場合に，リセットをかけたりするような場合がこれに当たります．この場合トラップしたソフトは，受信したキーのスキャンコードを把握しないといけません．つまりICのデータを読み込む必要があります．そして，その内容をチェックし，必要なければ元のBIOSルーチンにキーコードの解析処理を任せるのです．

しかし，この読み込んだ時に/RDYに関する処理を行わなければいけません．そして読み込んだ後には/RDYによるキーボードの待ち状態は解除されてしまっていますから，BIOSルーチンを処理中に新たなキーコードの受信が発生しうるなど，不具合を招く危険性があります．

そこで，以前あるソフトでは再送機能をこの目的に使用していました．つまり，まず最初に自分がキーボードのデータを取得し，その時にキーボードがデータを再送するようにしておくのです．そして，2回目のデータ受信の時には無条件でBIOSに制御を渡すようにします．こうすれば，/RDYハンドシェークに関しての問題はなくなります．

しかし，このような処理は基本的に好ましいとはいえません．ほとんどあり得ないことですが，もしキーボードからのデータがほんとうに受信エラーを起こした場合に，このような制御をしているとパニック状態になります．さらに，同様な処理方法をする複数のトラップルーチンが重なっている場合に，どのような状況になるかというのはちょっと考えてみればわかることです．

この問題を避けるためには，シリアルICのRTSを利用する必要があります．この方法については後述しますが，しかしこのやり方でもリスクを完全に回避することはできません．いちばん完全なのは，BIOSをまったく使用せず，ドライバを自前で書いてしまうことです．

一般にキーボードに関して処理を追加する場合，キーボードドライバ自体を置き換えたくなる場合と，既存のドライバにほんの少しだけ処理を追加したい場合とが考えられます．前者はキーボードの配列をJISからASCIIなどに変更する場合などがあり，後者には前記のようにキーコンビネーションによりリセットをかけられるようにする場合，古いタイプのキーには用意されていないバッククォートを入力できるようにする場合などが考えられます．今回は後者のやり方について解説し，前者の方法については次回に回すことにします．キーBIOS自体の構造やワークエリアなどについても，次回に解説します．

キーボードロック/リセットプログラム KEYEXT.ASM

今回作成するプログラムのリストをKEYEXT.ASMに示します．このプログラムは，キー関係の処理の主要部分はBIOSに任せたまま，キーコンビネーションによるリセット機能と，キーロック機能を実現するものです．常駐形式のプログラムとして作成し，これを常駐させることによってキー入力機能を拡張します．常駐関係の処理プログラムRES3.ASMも掲載したのがかなり前になるので，今回は再掲してみることにします．

my_id

KEYEXT.ASMの先頭部分には，このRES3.ASMとリンクするためのシンボルが定義されています．my_idは，プログラム固有の値です．常駐しているプログラムとは，このキーワードを用いて交信します．

割り込み処理テーブル

次に，トラップする割り込み関係の処理テーブルがあります．今回は，キーボード割り込みであるINT09だけ

リスト2 keyext.asm

```
        dosseg
        .model  small,pascal
        masm51

        .stack  2000h

        .code
        public  my_id,interrupt_list,init_func,release_func
        public  check_func,intf_func

        include condx.inc

my_id   dw      2591h

;------------------------------
; interrupt interface
;

int_work        struc
int_num         dw      ?
int_func        dw      ?
int_old         dd      ?
int_work        ends

interrupt_list  label   byte
i09     int_work        <09h,XINT09,0>
        dw      0

release_func    proc
        ret
release_func    endp

intf_func       proc
        ret
intf_func       endp

;------------------------------
; key interrupt service routine
;

        .data
key_buffer_work dw      ?
system_mode     dw      0
last_key_pointer        equu    526h
key_count       equ     528h
key_map equ     52ah
key_shift       equ     53ah

IS_LOCKED_KEY   equ     1

        .code
XINT09:
        sti
        push    ds
        push    es
        push    ax
        push    bx
        mov     ax,_DATA
        mov     ds,ax
        mov     ax,0
        mov     es,ax

; key buffer work set
        mov     ax,es:[last_key_pointer]
        mov     key_buffer_work,ax

; fetch status and data
        ; waste time
        mov     al,30
        dec     al
        jnz     $-2

        ; get status and check it
        in      al,43H
        and     al,38H
        @if     nz
                ; receive error, wait
                jmp     $+2
                jmp     $+2
                jmp     $+2
                jmp     $+2

                ; set re-transmit status
                mov     al,14H
                out     43H,al

                ; waste time
                mov     al,20
                dec     al
                jnz     $-2

                ; receive data, and dispose
                in      al,41H

                ; send EOI to PIC
                mov     al,20H
                out     0,al

                ; and return
                pop     bx
                pop     ax
                pop     es
                pop     ds
                iret
        @endif
        jmp     $+2
        jmp     $+2
        jmp     $+2
        jmp     $+2

        ; set RTS
        mov     al,36H
        out     43H,al

        ; wait
        mov     al,30
        dec     al
        jnz     $-2

        ; and receive data
        in      al,41H

        ; save char buffer size
        mov     bh,es:[key_count]

        ; check stop key
        cmp     al,60h
        @if     z
                ; stop key
                or      byte ptr es:[key_map+0ch],1

                ; send EOI
                mov     al,20h
                out     0,al

                ; now resume keyboard transmittion
                mov     al,16h
                out     43h,al

                test    system_mode,IS_LOCKED_KEY
                pop     bx
                pop     ax
                pop     es
                pop     ds
                @if     z
                        ; invoke int6 if key is not locked
                        int     6
                @endif
                iret
        @endif

        ; check copy key
        cmp     al,61h
        @if     z
                ; copy key
                or      byte ptr es:[key_map+0ch],2

                ; send EOI
                mov     al,20h
                out     0,al

                ; now resume keyboard transmittion
                mov     al,16h
                out     43h,al

                test    system_mode,IS_LOCKED_KEY
                pop     bx
                pop     ax
                pop     es
                pop     ds
                @if     z
                        ; invoke int5 if keyboard is not locked
                        int     5
                @endif
                iret
        @endif

        ; check special key conbination
        mov     ah,es:[key_shift]
        and     ah,18h          ; ctrl+graph
        cmp     ah,18h
        @if     z
                ; check del
                cmp     al,39h
                @if     z
                        ; invoke reset, check cpu type
                        push    sp
                        pop     ax
                        cmp     ax,sp
                        @if     nz
                                ; not 286/386 cpu
                                jmp     cs:dword ptr reset_vector
                                reset_vector    dd      0fffffff0h
                        @else
                                ; set shutdown port
                                in      al,35h
                                or      al,0a0h
                                mov     ah,10
                                dec     ah
                                jnz     $-2
                                out     35h,al

                                ; reset
                                mov     dx,1f0h
                                out     dx,al
                                jmp     $+0
                        @endif
                @endif

                ; check f8
                cmp     al,69h
                @if     z
                        ; set bit map
                        or      byte ptr es:[key_map+0dh],2

                        ; invert lock flag
                        xor     system_mode,IS_LOCKED_KEY

                        ; short cut
                        mov     al,20h
                        out     0,al
                        mov     al,16h
                        out     43h,al
                        pop     bx
                        pop     ax
                        pop     es
                        pop     ds
                        iret
                @endif
        @endif
```

ワンラインNEWS
日本電気は「PC-VAN」を韓国最大のパソコン通信網「PC-SERVE」から接続できるようにした。PC-VANからPC-SERVEを利用できるのはまだ1，2年かかる。

```

        ; chain interrupt
        cli
        pushf
        call    l09.int_old

        ; check lock flag
        test    system_mode,IS_LOCKED_KEY
        @if     nz
                ; resume pointer and key count
                mov     es:[key_count],bh
                mov     ax,key_buffer_work
                mov     es:[last_key_pointer],ax
        @endif

        pop     bx
        pop     ax
        pop     es
        pop     ds
        iret

init_seg        segment 'stack'
        assume  cs:init_seg

check_func      proc    far
        xor     ax,ax
        ret
check_func      endp

init_func       proc    far
        ret
init_func       endp

init_seg        ends
        end

```

リスト3　res3.asm

```
;-----------------------------
; res3: resident handling part
;
; $Log: RCS/res3.asm $
; revision 1.1 TAN 91/10/17 04:58:37
; Initial revision
;
;
;-----------------------------

        dosseg
        .model  small,pascal

        masm51                          ; delete if masm

        include condx.inc

        include utils.inc
        include dosfile.inc

        .code
        extrn   my_id:word
        extrn   interrupt_list:byte
        extrn   init_func:far
        extrn   release_func:near
        extrn   check_func:far
        extrn   intf_func:near

int_work        struc
int_num         dw      ?
int_func        dw      ?
int_old         dd      ?
int_work        ends

;-----------------------------
; int2f: mulitiplexed interrupt entry
; argument:
;       AX: ID
;       BX: function code
;          0 - check residence
;          1 - release program
; return:
;       DX: psp of program (function 0/1)
;       AX: broken

        .code
vect2f  dd      ?

_int2f:
        ; first, check interface ID
        cmp     ax,cs:my_id
        @if     nz
                jmp     vect2f          ; chain to old interrupt
        @endif

        sti                             ; enable interrupt
        push    ds
        mov     ax,_DATA                ; set data segment value
        mov     ds,ax

        ; branch with function code
        or      bx,bx
        @if     z
                ; check residence, return psp
                mov     dx,$psp         ; $psp is a system variable
        @else
        dec     bx
        @e_if   z
                ; release program
                call    release_func
                push    ds

                ; resume all vector
                mov     bx,offset interrupt_list
                @begin
                        mov     ax,cs:[bx].int_num
                        lds     dx,cs:[bx].int_old
                        add     bx,size int_work

                        or      ax,ax
                @while  nz
                        dos     <25h,,>
                @repeat

                ; release int2f vector
                lds     dx,vect2f
                mov     al,2fh
                dos     <25h,,>

                pop     ds

                ; then free environment
                mov     es,$psp         ; first, get environ seg
                mov     ax,es:[2ch]
                mov     bx,ax
                dec     ax
                mov     es,ax
                mov     ax,es:[1]       ; check owner of environ seg
                cmp     ax,$psp
                @if     z
                        ; environment is not released
                        mov     es,bx
                        dos     <49h,,>         ; free environ
                @endif

                ; return psp to release me
                mov     dx,$psp         ; $psp is a system variable
        @else
                call    intf_func       ; extended function
        @endif

        pop     ds                      ; resume ds value
        iret

;-----------------------------
; initialize routines
; define intialize code segment
;

        .data
end_pos db      0                       ; show the end of data segment
        public  end_pos

r_option        db      '-r'            ; option string definition
dup_message     db      'program is already installed',0dh,0ah,0
not_res_message db      'program not installed',0dh,0ah,0
rel_ok_message  db      'program correctly released',0dh,0ah,0
correct_message db      'program correctly installed',0dh,0ah,0
arg_err_message db      'argument range error',0dh,0ah,0
vect23  dd      ?                       ; for ctrl_c vector

init_seg        segment 'stack'         ; special segment for initialize
        assume  cs:init_seg             ; cs: init_seg

;-----------------------------
; strncmp: compare two strings
; argument:
;       p1, p2: buffer address
;       n: number to compare
; return:
;       ax: result, 0 for equall, nz for not equall

func    strncmp <p1, p2, n> <si di>
        mov     dx,0                    ; dx for flag
        mov     si,p1
        mov     di,p2
        push    ds                      ; setup es
        pop     es

        @begin
                cmp     n,0
        @while  nz
                lodsb                   ; compare characters
                scasb
                @if     nz
                        mov     dx,1    ; non zero value
                        @break
                @endif

                or      al,al           ; check string tail
                @if     z
                        @break
                @endif
                dec     n               ; decriment counter
        @repeat

        mov     ax,dx                   ; set return value
        ret
endf

;-----------------------------
; teminate: terminate function
; argument:
;       status: exit status
;
func    terminate       <status>
        ; restore int23 vector
        lds     dx,vect23
        mov     al,23h
        dos     <25h,,>

        ; terminate
```

をトラップするので，このための記述のみ用意してあります．さらにその後に，RES3から呼び出されるルーチンの定義がありますが，ここでは特別な処理を必要としないので，単なる空関数になっています．

key_buffer_work/system_mode

その次に，ローカルワークと定数が定義されています．key_buffer_workは，キーBIOSで管理されているリングバッファの最後端を格納しておきます．これを用いることにより，キーBIOSが呼び出された後，新たなキーコードがバッファ内に生成されたかどうかを知ることができます．system_modeは汎用のフラグで，ここではキーボードがロック状態にあることを示すために使用しています．そのほかの定数は，キーBIOSのワークエリアのアドレスを示すもので，内容は個別に説明します．

XINT09

XINT09が今回のプログラムのメイン部分で，キーボード割り込みを処理するものです．この割り込みがかかると，まず割り込みを許可してからレジスタを保存し，セグメントレジスタをセットアップします．ここではDSをローカルなデータセグメントに設定し，ESレジスタはBIOSのワークエリアをアクセスするために，0に設定します．

次に，前記のようにBIOSのポインタをローカルワークに保存し，それからシリアルICのアクセスに行きます．データの引き取りに際して，十分なディレイを取る必要があることから，ダミーループを回して，時間を潰します．

```
        mov     ax,status
        dos     <4ch,,,>
endf

dummy_int:
        iret

;──────────────────────────
; initial: initialize part
;

initial:
        @setup                          ; parse command line

        ; disable ^c interrupt
        mov     al,23h                  ; get ^c vector
        dos     <35h,,,>
        mov     word ptr vect23,bx      ; save it
        mov     word ptr vect23+2,es

        push    ds                      ; set dummy interrupt entry
        push    cs
        pop     ds
        mov     dx,offset dummy_int
        dos     <25h,,,>
        pop     ds

        push    ds
        mov     cx,_TEXT
        mov     ds,cx
        assume  ds:_TEXT
        mov     ax,my_id                ; then check residence
        pop     ds
        assume  ds:_DATA

        mov     bx,0
        mov     dx,bx
        int     2fh

        ; duplicate check
        or      dx,dx
        @if     nz                      ; already installed
                ; check R option
                mov     bx,_argv
                strncmp <offset r_option,[bx+2],2>      ; compare option
                @if     nz              ; reinstall error
                        mov     dx,offset dup_message
                        @puts <dx>

                        terminate <1>   ; error code
                @endif

                ; release program
                push    ds
                mov     cx,_TEXT
                mov     ds,cx
                assume  ds:_TEXT
                mov     ax,my_id
                pop     ds
                assume  ds:_DATA
                mov     bx,1
                int     2fh

                ; and release program area
                mov     es,dx
                dos     <49h,,,>

                ; message
                mov     dx,offset rel_ok_message
                @puts   <dx>

                ; terminate successfully
                terminate <0>
        @endif

        ; check -r flag
        mov     bx,_argv
        strncmp <offset r_option,[bx+2],2>
        @if     z
                mov     dx,offset not_res_message       ; not yet installed
                @puts   <dx>

                terminate <1>           ; error return
        @endif

        ; check condition
        call    check_func
        or      ax,ax
        @if     nz
                terminate <1>           ; error
        @endif

        ; install vector
        push    ds
        mov     cx,_TEXT
        mov     ds,cx
        assume  ds:_TEXT

        mov     al,2fh                  ; save 2fh vector
        dos     <35h,,,>
        mov     word ptr vect2f,bx
        mov     word ptr vect2f+2,es

        mov     dx,_TEXT                ; set 2fh vector
        mov     ds,dx
        mov     dx,offset _int2f
        dos     <25h,,,>

        ; and install other interrupts
        mov     si,offset interrupt_list
        @begin
                mov     ax,[si].int_num
                or      ax,ax
        @while  nz
                dos     <35h,,,>
                mov     word ptr [si].int_old,bx
                mov     word ptr [si].int_old+2,es

                mov     dx,[si].int_func
                dos     <25h,,,>

                add     si,size int_work
        @repeat

        pop     ds
        assume  ds:_DATA


        ; message output
        mov     dx,offset correct_message
        @puts <dx>

        ; call sub function
        call    init_func

        ; stay resident
        mov     dx,offset dgroup:end_pos        ; calculate paragraph size
        add     dx,15                   ; data segment length
        mov     cl,4
        shr     dx,cl

        add     dx,_DATA                ; add data segment offset
        sub     dx,$psp

        mov     al,0                    ; success flag
        dos     <31h,,,>                ; keep process

init_seg        ends

        end     initial

```


ワンラインNEWS
ワイ・イー・データはFDDの海外生産を拡充する．6月からタイの現地メーカに生産委託するほか，英国現地法人の生産規模も引き上げる．

それから，シリアルICのステータスレジスタを読みに行き，受信エラーチェックをします．もし万一エラーが発生すれば，この割り込み処理ルーチンでローカルに再送要求を出します．エラーの場合にはBIOSの処理ルーチンに流す方法も考えられますが，まあローカルに処理するほうが無難であるといえるでしょう．BIOSではエラーの再送回数を制限していますが，実際問題としてそれほどのエラーが生じることは考え難いので，ここでは回数制限をせずエラーが発生する限り再送要求を出します．

再送要求を出すためには，プログラムのようにDTRのレベルをセットし，データを引き取ります．DTRの設定コマンドコードは14Hで，これによりエラーフラグのクリアとDTRの0レベルセットが行われます．さらに時間を潰した後，シリアルICのデータを引き取り，キーインターフェースの/RDYシグナルを降ろします．そして最後にPICにEOIコマンドを送出します．しばらくすると，再びキーボードからキーコードが送出されることになります．

エラーがなかった場合は，RTSとDTRを両方セットした状態に設定します．RTSがセットされていれば，シリアルICからデータを引き取っても，/RDYシグナルが降りません．それから，BIOSのワークエリアに格納されている，キーカウントを取得します．これはキーボードロックのために使用します．

[STOP][COPY]キー処理

次に，[STOP] [COPY] キーの処理を行います．これらも基本的にはBIOSに流してしまってもよいのですが，キーボードロック機構の関係から，内部的に処理しています．これらのキーが押されると，INT6/5のソフトウェア割り込みが発生し，これの処理ルーチンが^Cの処理をしたり，画面コピー処理をしたりするわけですが，キーボードロック時にはこの割り込みの発生自体も抑制したいわけです．

もう1つの理由は，これらの割り込みが発生した時点で，スタック上に直接の戻りアドレスが積まれていることを保証するためです．ある種の常駐型ソフトウェアやデバッガなどでは，INT6などが発生したときにスタックに積まれているアドレスが，表で実行されていたプログラムのアドレスと解釈するものがあります．BIOSを割り込みエミュレーションで呼び出していたりすると，このドライバ内部が中断アドレスとして理解されてしまいます．

これらの処理内容はほぼ同等であり，まずBIOSのキービットマップテーブルにおける，当該キーのフラグをセットします．それからEOIを送出し，次にRTSをリセットします．この時点で/RDYが落ちるので，キーボードは新たなデータを送出することが可能となります．最後にレジスタを復元した後ロックがかかっていなければ対応する割り込みを起動します．

処理アルゴリズムからわかるように，これらのソフトウェア割り込みに関しては再入に対する配慮はまったくありません．これはBIOSルーチンでも同様です．ですから [STOP] キーを押し続けると，INT6の処理ルーチンが時間のかかるものであった場合，INT6の再入が発生する可能性があります．[STOP]キーによって処理をするプログラムを記述する場合には，この再入に対する配慮をする必要があります．

[STOP] [COPY]キーが押されてなければ，このドライバでサポートする特殊キーコンビネーションをチェックします．まずBIOSのシフトキーワークエリアを参照し，[CTRL]キーと[GRAPH] キーが同時に押されていることをチェックします．これらが押されていれば，キーボードから受信し

たスキャンコードをチェックし，これが［DEL］または［F8］キーであるかどうかチェックします．

［DEL］キーのコードであれば，リセット処理をします．まずSPレジスタがPUSHされた時の値をチェックし，CPUがV30か80286以上かを調べます．V30CPUであれば，ベクタ参照でアドレスFFFF:0000にジャンプします．これでリセットがかかります．

80286以上のCPUであれば，まずシステムポートのSHUTDOWNビットに値をセットします．それからポート1F0HにOUTしてリセットをかけます．0F0HにOUTしてもよいのですが，仮想86EMSなどを使用している場合に，0F0Hへの出力がプロテクトされている可能性があります．場合によっては1F0Hに出力することでこれをごまかすことができます．NESAなどのシステムでも1F0Hへの出力でリセットがかかるかどうかについてはチェックしていません．

次に［F8］であるかどうかを調べます．［F8］であれば，やはりキービットマップをアップデートした後，システムのロックフラグを反転します．それから［STOP］キーなどの場合と同様に割り込みをショートカットして終了します．

いずれでもなければ，BIOS処理ルーチンにチェインします．最後にキーのロックフラグを調べ，これがセットされていればBIOSの中のワークエリアをリセットすることで，もし新しいキーコードが生成されていても，これをクリアしてしまいます．

プログラムの最後には，常駐時の初期化に関するセグメントとその処理ルーチン（ダミー）が書かれています．

◆プログラムの作成

完成したプログラムにするには，次のようにKEYEXT.ASMとRES3.ASMをアセンブルした後リンクします．

```
tasm   keyext
tasm   res3
tlink  keyext+res3
```

作成されたKEYEXT.EXEを実行することでキーボード処理拡張ルーチンが常駐し，［CTRL］+［GRAPH］+［DEL］キーコンビネーションによるソフトウェアリセットと，［CTRL］+［GRAPH］+［F8］コンビネーションによるキーボードロック（オルタネート）が実現されます．KEYEXT -rとすることで，常駐解除できます．

最後に，今回の処理でも残っているリスクについて解説します．BIOSの処理ルーチンに入った時点では，RTSがセットされているのでキーボードからのデータは送出されません．しかし，いずれBIOS自身がデータを取得する前に，シリアルICにDTRをセットするコマンドを送出します．このコマンドによって，RTSがリセットされてしまいます．

ですから，このコマンドが送出されてからシリアルICのデータを取得する前に，よりレベルが高いタイマ割り込みがかかり，その処理に時間が掛かっている間にキーボードが次のキーコードを送信してくると，先程のデータを取りこぼすことになってしまいます．実際にはほとんど考えなくてよいリスクといえますが，完全無欠な処理方法でないということは念頭において置く必要があります．キーボード処理ドライバを全部自前で書けば，手間はかかりますがこのようなリスクはなくなります．

短期集中連載

酒井法雄

WordBASICを使おう
――マクロの概要・基礎編――

Microsoft Windowsでのプログラミングというと，Microsoft C+SDKやBORLAND C++，Microsoft VisualBasicなどを思い出します．しかし，いずれもWindowsならではの難しい概念を覚えないといけません．もっと手軽に特定の用途に活用できるのが，Microsoft Wordのマクロ言語であるWordBASICです．ここでは，WordBASICの概要を紹介します．

Microsoft Wordの逆襲

わが国におけるパソコン上のワードプロセッサといえば，一太郎と相場が決まっていました．一太郎は歴史のあるワードプロセッサです．操作性が悪いだの遅いだの邪悪だのといわれながらも，ユーザーを増やしシェアを増やしてきました．また，プログラム自体の大きさも巨大化してきました．雑誌のワードプロセッサ売り上げランキングを見ても，いつでも一太郎はトップでした．

しかし，ここ数か月，ちょっとした異変が起きています．Microsoft Windows用のワードプロセッサであるMicrosoft Word for Windowsが発売と同時にトップに踊り出ました．その後，トップはキープしていないものの，つねに上位にランキングされています．これは，どうしたことでしょうか．

MS-DOS上でワードプロセッサを作るとなると，速度の関係もあってテキストVRAMを使うのが常套手段でした．しかし，これでは倍角やルビなど大きさの違うフォントを使うときにどうしても無理が出てしまいます．また，イタリックやボールドといった形の違うフォントの表現力にも限界があります．そこで，見た通りのできあがりを実現するために，グラフィックVRAMを使うことが考えられました．しかし，これも速度を考えると適切な手段ではありません．したがって，現在のワードプロセッサはテキストとグラフィックの両方のVRAMを使うことができるという二重構造になっています．

ご存じのように，Microsoft WindowsはGUI（Graphical User Interface）を実現したOSです．最初から使うのはグラフィックVRAMです．もちろん，速度はあまり速くありません．しかし，Windowsを使うために高速なCPUと大量のメモリを装備したマシンが普及しはじめています．

このような環境であれば，いまさらMS-DOSを使うまでもなく，最初からGUIを実現しているWindows上のワードプロセッサでいいわけです．

しかも，Microsoft Wordは米国でも高い評価を受けているワードプロセッサです．Microsoft Windows以外にMacintosh用もあり，世界的にはポピュラーなワードプロセッサなのです．それを日本語化したのです．もちろん，日本語を使うためにスペックを検討し，それを実現するのはたやすいことではなかったでしょう．しかし，何もないところからいきなりWindows用のワードプロセッサを作ったわけではないので，基本的な素性はたいへんよいものであったわけです．

わが国でのWindows用のアプリケーションは，まだまだ少ない状況です．「とりあえずWindowsは買ったがアプリがない」といったユーザーが飛びついたということも，もちろんあったでしょう．

このようなわけで，このままMS-DOSからWindowsの世界へと時代が移行すれば，わが国でのWindows用のワードプロセッサといえばMicrosoft Wordという時代になることでしょう．

もっとも，現在のわが国の標準WindowsマシンであるPC-9801のように画面が狭くて遅くて高いマシンでは，現実的にWindowsを使う気になれません．せめて，H98シリーズ程度のレゾリューションが欲しいところです．

Microsoft Wordの概要

このように，将来のわが国の標準ワードプロセッサと嘱望されているMicrosoft Wordですが，さきほども述べたように高い評価を受けているのには，もちろんそれなりの理由があります．

GUIといえば，一時はDTP (Desk Top Publishing) のためのものであるといった理解の方が多かったようですが，それだけではないことは，賢明な読者の皆様にはあえて説明する必要はないと思います．しかし，ワードプロセッサが日常的に使われている現在，次にくるものがDTPであることは，ワードプロセッサ専用機の世界を見ても明らかです．それには，各種のフォントばかりではなく，ビットマップファイルなどのグラフィックデータや表計算ソフトなどのデータとの有機的な結合が必要です．

Microsoft Wordはあくまでもワードプロセッサではありますが，かなりDTPを意識したものです．ある意味では，DTPといってもなんら差し支えないレベルになっています．その要因のひとつとしては，Microsoft WindowsというOSの持っているDDEなどの機能を満足していることがあげられます．たとえば，クリップボードを介しての，ほかのアプリケーションとのテキストやグラフィックデータのやりとり，DDEを使ってのMicrosoft Excelなどのアプリケーションとの通信といった機能があります．Microsoft Word単体ではなく，Windows上のほかのアプリケーションをトータルな環境として有機的に使うことができるのが，Microsoft Wordの強みなのです．

もちろん，ワードプロセッサとしての必要な機能はすべて揃っているといっていいでしょう．今までのMS-DOS上でのワードプロセッサは，実に多くの機能を実現するために多くのメニュー階層を使っていました．目的の機能を探し出すまでの手間が実に大きかったのです．これでは，新しく出た高機能なワードプロセッサがあっても，そうそう乗り換えるわけにはいきません．それゆえ，操作性はよくなくても一度操作を覚えた一太郎をいつまでも使うユーザーが多かったのです．

WordBASICとは

Microsoft Wordでは，基本的な機能だけの「ショートメニュー」と多彩な機能を持つ「フルメニュー」の使い分けができます．「フルメニュー」で拡張された機能の中には，マクロとして内蔵されているものもあります．

マクロとは，ある機能を実現する手順をなんらかの形で記憶させるものです．MIFESやVzなどのエディタには，キーストロークを記憶させてそれを何度でも再生できるキーボードマクロがあります．また，言語仕様と基本的関数を組み合わせて，プログラミング言語のように記述して，ある機能を実現するマクロ言語があります．Vzのマクロ言語は実に強力で，通信ソフトやゲームを作れるほどです．

Wordでは，キーボードマクロとマクロ言語の両方を使うことができます．このうち，マクロ言語はMS-DOS上のQuickBASICによく似た言語体系を持ち，WordBASICと呼ばれています．このWordBASICの関数やステートメントには，単語の削除や改ページといったワードプロセッサとしての基本的な機能が含まれています．これらのひとつひとつの関数やステートメントを呼び出せば，通常のワードプロセッサの機能を使うことができるわけです．

また，WordBASICの関数やステートメントには，システムラインを使った入出力やダイアログボックスを使った入出力，ファイルのオープンなどと

いったものも含まれています．これらをQuickBASICのような制御構文を使って記述すれば，独自の機能をあたかもWord内蔵の機能のように，Wordに追加することができるわけです．

Microsoft Wordに含まれてくる「macros. doc」には，文字種の変換やスペルチェッカ・英語類義語辞典のメニューへの追加，テキストファイルコンバータなどのマクロが記述されており，自在に追加することができるようになっています．

WordBASICの環境

本来はMicrosoft Wordはワードプロセッサですから，そこで編集できるファイルは基本的には文書ファイルです．Microsoft Wordからマクロ言語であるWordBASICを使うには，「補助(U)」メニューにあるマクロ関係のメニューを使います．このとき，マクロ関係のメニューは，「フルメニュー」にしていないと出てこないので，注意が必要です．

マクロ関係のメニューには，次のものがあります（写真1）．

1.マクロ記録
2.マクロ実行
3.マクロ編集
4.キー定義
5.メニュー定義

「マクロ記録」は，いわゆるキーボードマクロの記録です．このときマクロの名前を決めます．あとは，やりたいキー操作をしたあと，もう一度「補助（U)」メニューを開くと，さきほどまで「マクロ記録」になっていたところが「マクロ記録終了」になっていますから，それを選択します．

「マクロ実行」は，マクロを選択して実行するメニューです．

「マクロ編集」は，WordBASICを使うときに一番使われるものでしょう．このメニューを選択すると，編集するマクロ名を指定するダイアログが開きます．ここで，既存のマクロを選択すると，そのマクロを編集することができます．新しいマクロ名を指定すると，QuickBASICのように，「Sub MAIN」と「End Sub」があらかじめ書かれたウィンドウが現れます．ここに，マクロを記述します．

また，このときメニューバーの下にある文書関係の情報を表示しているバーが変更され，「開始」，「ステップ」，「ステップSUB」，「トレース」，「変数」のプッシュボタンのあるバーが表示されます（写真2）．これらのボタンは，WordBASICで作成したマクロをデバッグするときに使います．それぞれのボタンは，次のような意味をもちます．

・開始

マクロを実行します．「ステップ」や「ステップSUB」でマクロが停止しているときには，「継続」という表示になります．

・ステップ

1ステップずつ実行します．実行する命令がサブルーチンにあるときにも，サブルーチンの命令が1ステップずつ実行されます．

・ステップSUB

1ステップずつ実行します．実行する命令がサブルーチンにあるときには，サブルーチンを1ステップとして実行されます．

・トレース

編集中のマクロを実行し，実行中の行を反転表示します．

写真1　マクロ関連メニュー

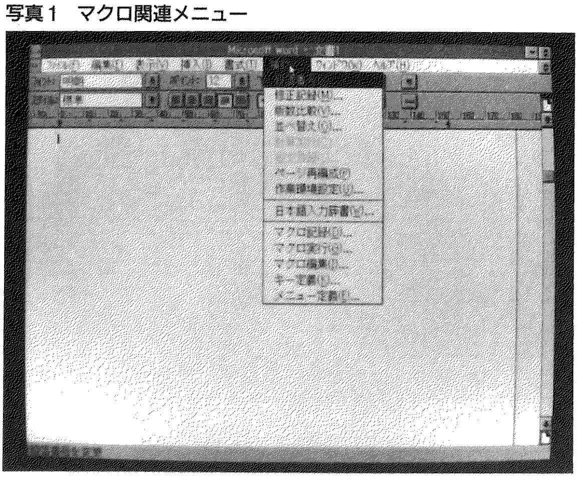
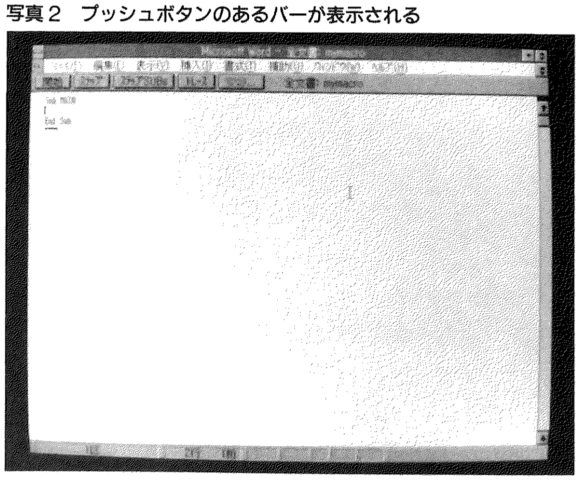


写真2　プッシュボタンのあるバーが表示される



・変数

変数の一覧や内容を表示できます．変数の内容は変更することもできます．

「キー定義」は，現存するマクロを[CTRL]+[キー]のような形ですぐに呼び出せるように定義するものです．

「メニュー定義」は，現存するマクロをメニューに追加するものです．

このように，WordBASICで定義したマクロは，コントロールキーを使った定義だけでなく，メニューにも手軽に追加することができます．また，完璧とはいえないまでも，デバッグ環境を装備しているので，デバッグも今までのいわゆるマクロ言語とは比べものにならないほどラクでしょう．

WordBASICの言語仕様

それでは，実際にWordBASICの言語仕様を紹介しましょう．

1.データ型

WordBASICのデータ型には，次の3つのみがあります．

・文字列型

文字列型は，変数名の後ろにドル記号（$）がついたものです．通常のBASICと同じですが，DIMを使った宣言はできません．文字列の長さはメモリに依存しますが，最大で32KBまで使うことができます．

・数値型

倍精度不動小数点型だけが使えます．QuickBASICなどに慣れていると，「DIM i AS INTEGER」などとつい打ってしまいますが，整数型は使えませんので注意してください．

・ダイアログレコード

ダイアログの各フィールドに対応する値をもつ複合型のデータ型です．詳細は，後に述べます．この型を使うときには，リスト1のように宣言します．

これらの変数は，基本的にはローカル変数です．SubやFunctionなどの中でのみ有効ですが，グローバル変数として使いたいときには，SubやFunctionの外に，リスト2のように「Shared」キーワードを使って宣言します．このあたりは，QuickBASICとは違っていますので，注意してください．

2.式

通常のBASICで使われるような式や演算子をWordBASICでも使うことができます．

WordBASICの論理式は，Trueは-1，Falseは0を返します．

ビット演算子は，一度整数に変換されてからビット処理されます．

演算子の優先順位は，乗除が加減よりも優先されます．括弧を使って優先順位を変えることも可能です．表1にWordBASICの演算子の優先順位を示します．

このあたりは，通常のBASICと同じですから，あまり気にする必要はないでしょう．

3.ステートメントと関数

WordBASICはQuickBASICによく似ています．もちろん，ステートメントと関数もあります．一般には，戻り値のないものをステートメント，後ろに括弧がついていて戻り値のあるものを関数と呼んでいます．

WordBASICには，BASIC本来のステートメントと関数のほかに，WordBASICに特有のステートメントと関数と，ダイアログ定義ステートメントがあります．

ステートメントと関数については，後に詳しく述べます．

4.制御構造

WordBASICは，構造化制御構文をサポートしたQuickBASICの制御構文ときわめて良く似ています．WordBASICで使える制御構文を，表2にあげておきます．

リスト1

```
DIM DialogRecord As DialogBox
```

リスト2

```
Dim Shared i

Function Test$
        i = 3
        Test$ = Str$(i)
End Function

Sub MAIN
        Print Test$; i + 1
End Sub

-----------------------------------------------------------------
実行結果
 3 4
```

QuickBASICと比較すると,「DO...LOOP」文がサポートされていませんが,通常の使用には「While...Wend」で十分でしょう.これでもキチンと構造化して書くことができますから,イカすプログラミングも可能でしょう.

5.サブルーチン

ユーザー定義のステートメントともいえるのがサブルーチンです.QuickBASICでは「GOSUB」と「RETURN」で定義された従来からあるグローバルなサブルーチンと区別するために,「SUB...END SUB」で定義されたものを「サブプログラム」と呼んでいました.WordBASICで使えるサブルーチンは,このサブプログラムと同じものです.

もちろん,WordBASICサブルーチンでも,引数をとることができます.

6.ユーザー定義関数

ユーザー定義の関数は,QuickBASICでも2つの方法がありました.「DEFFN」キーワードを使って1行で定義するものと,「FUNCTION...END FUNCTION」で自在に書くことができるものです.

WordBASICでのユーザー定義関数は,「Function...End Function」で囲まれた部分に記述するもので,戻り値があることを除けば「Sub...End Sub」ときわめて良く似ています.引数をとって戻り値を設定することも可能です.

7.入出力

WordBASICはWindows用のアプリケーションであるMicrosoft Word上で動作するわけですから,Windows独自のインターフェイスとWord独自のインターフェイスを満足する必要があります.そこで,ディスプレイやキーボード,マウスとの入出力には独自の関数やステートメントが用意されています.また,従来のBASICにあった関数やステートメントと同じ名前でも,動作が違ってきます.

次に,WordBASICで使える入出力ステートメントと関数を列挙します.

・Input

通常のBASICのINPUT文と似ていますが,入出力は画面下部のステータスバーで行われます(写真3).

・InputBox$ ()

ダイアログボックスを表示し,〈了解〉ボタンが押されたときに,入力された文字列を返します(写真4).この関数は,リスト3の書式になります.

ダイアログのタイトルであるTitle$と入力のデフォルト値Default$は,省略可能です.

・MsgBox

このステートメントは,指定した内容のボタンを押されるまで,メッセージや疑問符や感嘆符などのアイコンを表示するものです.

このステートメントは,リスト4の書式になります.

ダイアログのタイトルであるTitle$

表1　WordBASICの演算子の優先順位

−	負の符号として使用する
*	2つの数値を乗算する
/	最初の数値を2番目の数値で除算する
Mod	指定された2つの数値を整数に四捨五入し,最初の数値を2番目の数値で除算しその剰余を返す
+	2つの数値を加算する
−	最初の数値から2番目の数値を減算する
+	2つの文字列を連結する
=	2つの数値が等しいかどうかテストする
<>	2つの数値が等しくないかどうかテストする
>	最初の数値が2番目の数値より大きいかどうかテストする
<	最初の数値が2番目の数値より大きいかどうかテストする
<=	最初の数値が2番目の数値以下かどうかテストする
>=	最初の数値が2番目の数値以上かどうかテストする
Not	ビット演算を行って数値の補数を計算したり,論理式の結果であるTRUE,FALSEを逆に切り替える
And	ビット演算を行って数値の論理積を計算したり,2つの条件を比較して,それらがともにTRUEであるときTRUEを返す
Or	ビット演算を行って数値の論理和を計算したり,2つの条件を比較して,いずれかがTRUEであるとき,TRUEを返す

*優先順位の高い順に配列

表2　WordBASICで使える制御構文

```
For...Next
Goto
IF...Elself...Else...End If
On Error Goto
Select Case...End Select
Stop
While...Wend
```

とボタンの表示内容値Typeは，省略可能です．

・MsgBox ()

MsgBoxステートメントとほぼ同じですが，押されたボタンによって違う内容の数値を返す関数です（写真５）．

この関数は，リスト５の書式になります．

ダイアログのタイトルであるTitle$とボタンの表示内容値Typeは，省略可能です．

・Print

通常のBASICでのPRINT文と似ていますが，入出力は画面下部のステータスバーで行われます（写真６）．

8.ファイル入出力

WordBASICでのファイルは，シーケンシャルファイルのみを扱うことができます．ランダムアクセスファイルは使うことができません．

9.ブックマーク

WordBASICはワードプロセッサを効率良く使うためのマクロですから，文書中の位置などを示すブックマークをサポートしています．

ブックマークを表３にあげておきま

リスト３

```
A$ = InputBox$(Prompt$, [Title$], [Default$])
```

リスト４

```
MsgBox Message$, [Title$], [Type]
```

リスト５

```
Num = MsgBox(Message$, [Title$], [Type])
```

写真３　画面下部のステータスバーで入出力が行われる：Input

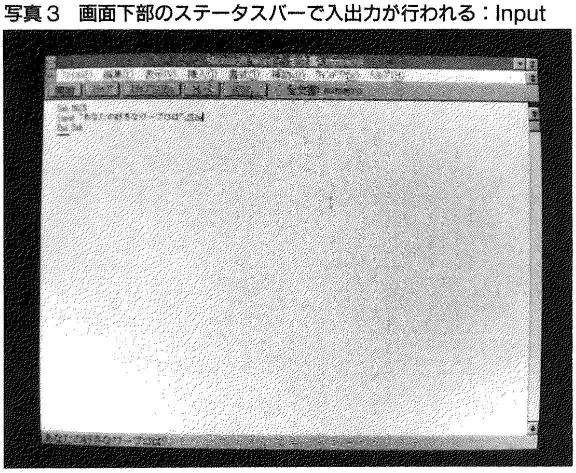


写真４　入力された文字列を返す

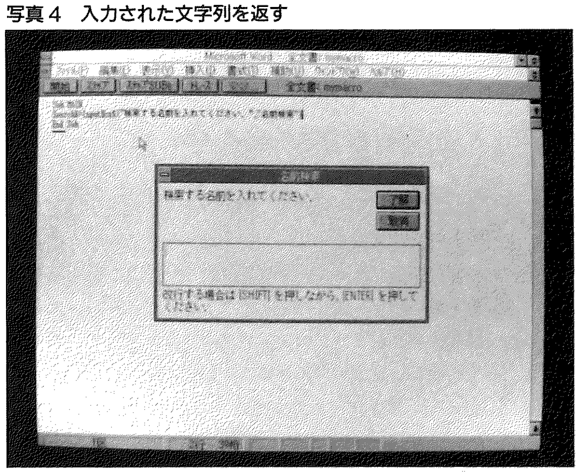


写真５　押されたボタンによって違う内容の数値を返す

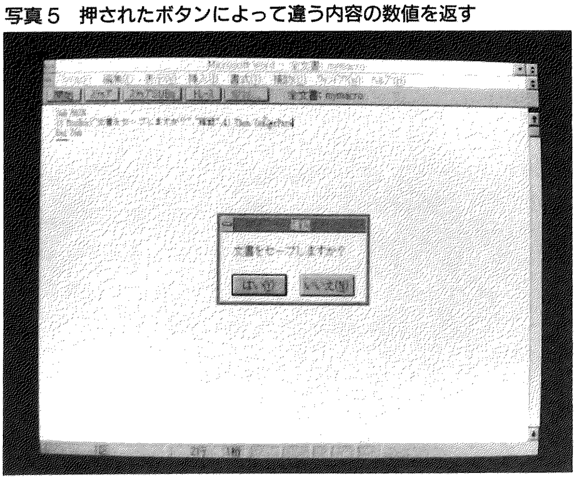
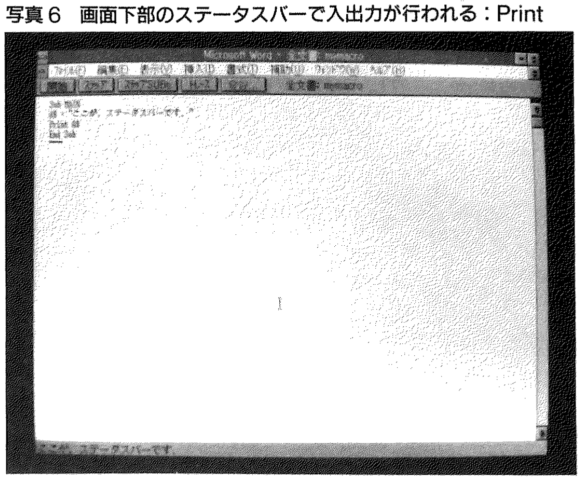


写真６　画面下部のステータスバーで入出力が行われる：Print



す.

10.DDE

Microsoft Wordでは，DDE (Dynamic Data Exchange) をサポートしています．もちろん，WordBASICでもDDEを使うことができます．

DDEとは，DDEの機能をサポートしているアプリケーション同志で通信をし，データを交換することです．

DDEを使うには，次の手順で作業します．

・DDE通信の開始

DDEInitiate関数を使って，DDEチャンネルをオープンし，チャンネル番号を取得します．

・データの取得

DDERequest$関数を使って，オープンしたチャンネルから情報を取得します．

・通信の終了

DDETerminateステートメントを使って，チャンネルをクローズします．

DDEでの通信の詳細は，次回に紹介します．

11.ダイアログボックスの作成

WordBASICでは，Windowsで使うことのできる通常のコントロールを使ったダイアログを自由に作成することができます．

ダイアログボックスを定義するには，「Begin Dialog」と「End Dialog」の間にダイアログの形状を示す内容を記述します（リスト6）．

ここに記述することのできるコントロールを表4にあげます．

こうして作成したダイアログボックスは，ダイアログボックスのデータ型を定義したにすぎません．実体を宣言するには，リスト7のように記述します．

これで，SampleDlg型のダイアログボックスをdlgという名前で宣言しました．このダイアログを画面に表示するには，リスト8のように記述します．

このように，WordBASICではダイアログボックスも1つのデータ型として扱われます．このほかにも，FileOpenなどのよく使われるダイアログボックスは，あらかじめ組み込まれており，WordBASICから使うことができます．

12.WordBASICの階層

WordBASICのマクロには，表5の3つの階層があります．

組み込みコマンドは，あらかじめWordに内蔵されているコマンドです．もし，組み込みコマンドと全文書とテンプレートの階層に同じ名前のマクロがあれば，テンプレートの階層のマクロが実行されます．テンプレート階層に同じ名前のマクロがなければ，全文書のマクロが実行されます．

つまり，組み込みコマンドと同じ名前のマクロを作っても，そっくり内容を入れ換えることもできますし，特定のテンプレートのときだけ入れ換えることもできるわけです．

WordBASICのステートメントと関数

WordBASICはMicrosoft Word

表3　WordBASICで使えるブックマーク

¥Sel	現在の選択範囲
¥PrevSel1	編集が行われる以前の選択範囲1（起動時には存在しない）
¥PrevSel2	編集が行われる以前の選択範囲2（　〃　）
¥StartOfSel	選択範囲の起点
¥EndOfSel	選択範囲の終点
¥Line	作業中の行（選択範囲の先頭行）
¥Char	作業中の文字（選択範囲の先頭文字）
¥Para	作業中の段落（選択範囲の先頭段落）
¥Section	作業中のセクション（選択範囲の先頭セクション）
¥Doc	文書全体
¥Page	作業中のページ
¥StartOfDoc	文書の先頭
¥EndOfDoc	文書の末尾
¥Cell	セル
¥Table	表組み
¥HeadingLevel	見出しレベル

表4　コントロール一覧

CheckBox	チェックボックス
OptionButton	オプションボタン
OptionGroup	オプションボタングループ
ListBox	リストボックス
ComboBox	コンボボックス
OKButton	了解ボタン
CancelButton	取消ボタン
Text	テキスト
TextBox	テキストボックス

のマクロ記述言語ですから，通常のBASICのステートメントや関数以外にも，ワードプロセッシングに役に立つものや，Windowsという環境ならではのためのものが用意されています．

とうてい全部についての詳細は紹介できませんので，どのようなステートメントや関数があるのかを表6にまとめました．名前から類推しても，だいたいの機能はわかると思います．詳細については，「Microsoft Word for Windows マクロ入門」(マイクロソフト株式会社監修／翔泳社刊／2,800円）をご覧ください．

今回は，WordBASICの概要についてのダイジェストでした．少々退屈な内容だったかもしれませんが，今までのBASICとはかなり違うことを理解していただけたことと思います．次回は，実際にWordBASICを使っていろいろなプログラミングにチャレンジしてみます．

リスト6

```
Begin Dialog SampleDlg 300, 50
    OKButton 105, 27, 88, 18
End Dialog
```

リスト7

```
Dim dlg As Dialog SampleDlg
```

リスト8

```
Dialog dlg
```

表5

テンプレート	特定のテンプレートのみで実行可能
全文書	すべての文書で実行可能
組み込みコマンド	あらかじめ組み込まれているコマンド

表6　ステートメントと関数

■ほかのBASICと共通のステートメントと関数

定義，宣言のためのステートメント
- Declare
- Dim
- Function...End Function
- Let
- Sub...End Sub
- Redim

プログラムの分岐，制御を行うステートメント
- Call
- For...Next
- Function...End Function
- Goto
- If...ElseIf...Else...End If
- On Error
- Select Case...Case...Case Else...End Select
- Stop
- Sub...End Sub
- While...Wend

ディスプレイ，キーボード入出力のためのステートメントと関数
- Input
- InputBox$ ()
- Line Input
- Print
- MsgBox
- MsfBox ()

文字列処理関数
- Asc ()
- CDbl$ ()
- CExt$ ()
- Chr$ ()
- CInput$ ()
- CInstr$ ()
- CLeft$ ()
- CLen ()
- CMid$ ()
- CPos ()
- CRight$ ()
- CSng$ ()
- InStr ()
- LCase$ ()
- Left$ ()
- Len ()
- Mid$ ()
- Right$ ()
- Str$ ()
- String$ ()
- UCase$ ()

数値処理関数
- Abs ()
- Int ()
- Sgn ()
- Rnd ()
- Val ()

エラー処理のためのステートメント
- Err

Error
On Error

シーケンシャルファイル，ディスク ファイルを操作するステートメントと関数
ChDir
CInput$ ()
Close
Eof ()
Files$ ()
Input
Input$ ()
Kill
Line Input
Lof ()
MkDir
Name
Open
Print
Read
RmDir
Seek
Seek ()
Shell
Write

そのほかのステートメントと関数
Beep
Date$ ()
Rem
Shell
Times$ ()

■WordBASICに特有のステートメントと関数

ブックマークを処理するためのステートメントと関数
BookmarkName$ ()
CmpBookmarks ()
CopyBookmark
CountBookmarks ()
EditGoto
EmptyBookmark ()
ExistingBookmark ()
GetBookmark$ ()
InsertBookmark
SetEndOfBookmark
SetStarOfBookmark

Wordの使用環境を設定するためのステートメントと関数
AppInfo$ ()
Cancel
ChangeRulerMode
ColumnSelect
CountFiles ()
CountWindows ()
Date$ ()
DisableInput
EditSummaryInfo
Extend Selection
FileName$ ()
GetProfilesString$ ()
IsDirty ()
OK
Overtype
Overtype ()
SetDirty
SetProfileString
Time$ ()
Window
WindowName$ ()

DDE, OLEに関連するステートメントと関数
DDEExcute
DDEInitiate ()
DDEPoke
DDERequest$ ()
DDETerminate
DDETerminateAll
EditLinks
EditObject
EditPaseteSpecial
InsertObeject

ダイアログボックスの定義と制御を行うステートメントと関数
Begin Dialog
CancelButton
CheckBox
ComboBox
Dialog
Dim
End Dialog
GetCurValues
GropBox
ListBox
OKButton
OptionButton
OptionGroup
Redim
Text
TextBox

文書編集のためのステートメントと関数
CopyFormat
CopyText
DeleteBackWord
DeleteWord
EditClear
EditCopy
EditCut
EditGoto
EditHeaderFooter
EditHeaderFooterLink
EditObject
EditPaste
EditPasteSpecial
EditRepeat
EditReplace
EditReplaceChar
EditReplacePara
EditSearch
EditSearchChar

EditSearchFound ()
EditSearchPara
EditSelectAll
EditTable
EditUndo
Insert
InsertFile
MoveText
OK
RepeatSearch
Selection$ ()
SelectTable
Spike
UnSpike

フィールドを操作するためのステートメントと関数

DoFieldClick
EditLinks
EditObject
InsertDateField
InsertField
InsertFieldChars
InsertFile
InsertIndex
InsertIndexEntry
InsertObject
InsertPageField
InsertPicture
InsertTableOfContents
InsertTimeField
LockFields
NextField
NextField ()
PrevField
PrevField ()
ToggleFieldDisplay
UnLinkFields
UnLockFields
UpdateFields
UpdateSource
ViewFieldCodes
ViewFieldCodes ()

ファイル操作，印刷のためのステートメントと関数

Activate
AppActivate
ChDir
CountFiles ()
DocClose
File1
File2
File3
FileClose
FileExit
FileFind
FileName$ ()
FileNew
FileOpen
FilePrint
FilePrinterSetup
FilePrintMerge
FilePrintPreview
FilePrintPreviewBoundaries
FilePrintPreviewPages
FilePrintPreviewVertical
FileSave
FileSaveAll
FileSaveAs
Files$ ()
InsertFile
Kill
MkDir
Name
RmDir
SaveTemplate
UpdateSource

書式設定のためのステートメントと関数

Bold
Bold ()
CenterPara
CenterPara ()
CountFonts ()
ChangeCase
CharColor
CharColor ()
CloseUpPara
CopyFormat
CountStyles ()
DistributePara
DistributePara ()
DoubleUnderline
DoubleUnderline ()
EditHeaderFooter
EditHeaderFooterLink
EditReplaceChar
EditReplacePara
EditSearchChar
EditSearchPara
Font
Font$ ()
FontSize
FontSize ()
FormatCharacter
FormatDefineStyles
FormatDefineStylesChar
FormatDefineStylesPara
FormatDefineStylePosition
FormatDefineStylesTabs
FormatDocument
FormatParagraph
FormatPicture
FormatPosition
FormatSection
FormatStyles
FormatTable
Format Tabs
GrowFont
HangingIndent
Hidden
Hidden ()
Indent

InsertBreak
InsertColumnBreak
InsertFootnote
InsertObject
InsertPageBreak
InsertPageNumbers
InsertPara
InsertPicture
InsertTable
InsertTableToText
Italic
Italic ()
JustifyPara
JustifyPara ()
LeftPara
LeftPara ()
NormalStyle
NormalStyle ()
OpenUpPara
ResetChar
ResetChar ()
ResetFootnoteContNotice
ResetFootnoteContSep
ResetFootnoteSep
ResetPara
ResetPara ()
RightPara
RightPara ()
SelType
SelType ()
ShrinkFont
SmallCaps
SmallCaps ()
SpacePara1
SpacePara1 ()
SpacePara2
SpacePara2 ()
SpacePara15
SpacePara15 ()
StyleName$ ()
SubScript
SubScript ()
SuperScript
SuperScript ()
TabLeader$ ()
TabType ()
Underline
Underline ()
Underlines
Underlines ()
UnHang
UnIndent
WordUnderline
WordUnderline ()

短文登録のためのステートメントと関数
CountGlossaries ()
EditGlossary
GetGlossary$ ()
GlossaryName$ ()
SetGlossary
Spike
UnSpike
UtilGlossary

ヘルプに関連するステートメントと関数
Help
HelpAbout
HelpActiveWindow
HelpContext
HelpIndex
HelpKeyboard
HelpUsingHelp

マクロの作成，編集を行うステートメントと関数
CountMacros ()
DisableInput
MacroAssignToKey
MacroAssignToMenu
MacroEdit
MacroName$ ()
MacroRecord
McroRun
OnTime
PauseRecorder
RecordNextCommand
ShowVars
Super

Wordの作業モードを設定するためのステートメントと関数
AppInfo$ ()
Cancel
ChangeRulerMode
ColumnSelect
CopyFormat
CopyText
IconBarMode
MenuMode
MoveText
OverType
OverType ()
RulerMode

アウトライン機能に関連するステートメントと関数
OutlineCollapse
OutlineDemote
OutlineExpand
OutlineLevel ()
OutlineMoveDown
OutlineMoveUp
OutlinePromote
OutlineShowFirstLineView
ShowAllHeadings
ShowHeading1
ShowHeading2
ShowHeading3
ShowHeading4
ShowHeading5
ShowHeading6
ShowHeading7
ShowHeading8
ShowHeading9

ワンラインNEWS
ソニーは本体重量が 4.5 キロと小型軽量なラップトップ型WS「NWS-3150」を発売。ラップトップ型としては業界最高水準の 37 MIPSの処理能力を持つ。

ViewOutline
ViewOutline ()

挿入ポインタの移動，選択のためのステートメントと関数
CharLeft
CharLeft ()
CharRight
CharRight ()
EditGoto
EditSelectAll
EndOfColumn
EndOfDocument
EndOfLine
EndOfRow
EndOfWindow
ExtendSelection
GoBack
Insert
LineDown
LineDown ()
LineUp
LineUp ()
NextCell
NextCell ()
NextField
NextField ()
NextObject
NextObject ()
NextPage
NextPage ()
NextTab ()
PageDown
PageDown ()
PageUp
PageUp ()
ParaDown
ParaDown ()
ParaUp
ParaUp ()
PrevCell
PrevCell ()
PrevField
PrevField ()
PrevObject
PrevObject ()
PrevPage
PrevPage ()
PrevTab ()
Selection$ ()
SelectTable
SentLeft
SentLeft ()
SentRight
SebtRight ()
StartOfColumn
StartOfDocument
StartOfLine
StartOfRow
WordOfWindow
WordLeft
WordLeft ()
WordRight
WordRight ()

表組みのためのステートメントと関数
EditTable
EndOfColumn
EndOfRow
FormatTable
InsertTable
InsertTableToText
NextCell
NextCell ()
PrevCell
PrevCell ()
SelectTable
StartOfColumn
StartOfRow

Wordの補助機能に関連するステートメントと関数
Activate
AppActivate
ControlRun
Declare
GetProfileString$ ()
RenameMenu
SendKeys
SetProfileString
UtilCalculate ()
UtilCompareVersions
UtilCustomize
UtilGetSpelling
UtilGetSpelling ()
UtilGetSynonyms
UtilGetSynonyms ()
UtilGrossary
UtilHyphenate
UtilRenumber
UtilRepaginateNow
UtilRevisionMarks
UtilSort
UtilSpelling
UtilSpellSelection
UtilThesaurus
UtilUpdateIMEDic
ViewFullMenus ()
ViewIMEBar
ViewIMEBar ()
ViewMenues ()
ViewRibbon
ViewRibbon ()
ViewRuler
ViewRuler ()
ViewShortMenus
ViewStatusBar
ViewStatusBar ()

画面表示のためのステートメントと関数
FilePrintPreview
FilePrintPreview ()
ShowAll
ShowAllHeadings

ワンラインNEWS
セイコーエプソンは米アップルコンピュータの「Macintosh」に対応したレーザープリンタ「LP 2000 LT」を6月末に発売する．

ShowHeading1
ShowHeading2
ShowHeading3
ShowHeading4
ShowHeading5
ShowHeading6
ShowHeading7
ShowHeading8
ShowHeading9
ToggleFieldDisplay
ViewAnnotations
ViewAnnotations ()
ViewDraft
ViewDraft ()
ViewFieldCodes
ViewFieldCodes ()
ViewFootnotes
ViewFootnotes ()
ViewFullMenus
ViewIMEBar
ViewIMEBar ()
ViewMenus ()
ViewOutline
ViewOutline ()
ViewPage
ViewPage ()
ViewPreferences
ViewRibbon
ViewRibbon ()
ViewRuler
ViewRuler ()
ViewShortMenus
ViewStatusBar
ViewStatusBar ()

ウィンドウ操作のためのステートメントと関数

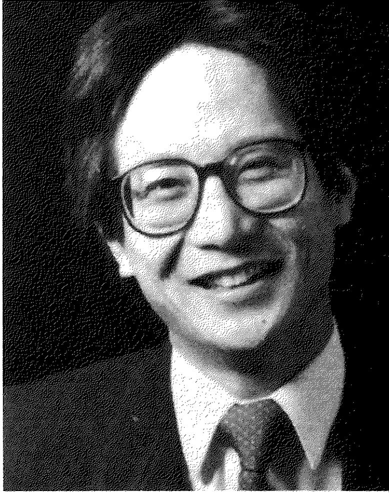
Activate
AppActivate
AppMaximize
AppMaximize ()
AppMinimize
AppMinimize ()
AppMove
AppRestore
AppSize
ClosePane
CountWindows ()
DocClose
DocMaximize
DocMaximize ()
DocMove
DocRestore
DocSize
DocSplit
DocSplit ()
EndOfWindow
FileClose
HLine
HPage
HScroll
HScroll ()
NextWindow
OtherPane
PrevWindow
StartOfWindow
VLine
VPge
VScroll
VScroll ()
Window ()
Window1
Window2
Window3
Window4
Window5
Window6
Window7
Window8
Window9
WindowArrangeAll
WindowName$ ()
WindowNewWindow
WindowPane ()

ワンラインNEWS
松下電器産業は電子ブックプレーヤー「KX-EBP I」を発売。従来，他メーカーは8ビットCPUを使っていたが，同社では16ビットCPUを採用，画面も大きくした。

SPECIAL INTERVIEW

オブジェクト指向プログラミングとアプリケーションフレームワークスがプログラマを解放する!

■ ■ ■ ボーランド社　Eugene Wang氏に聞く



Eugene Wang氏は，Borland InternationalのLanguage Business Unitを統括するvice president兼general managerである．The Borland International Developers Conferenceの会場において氏にインタビューする機会を得たので，ボーランドの言語製品における今後の戦略について語ってもらった．

Windowsへの移行とともにオブジェクト指向開発ツールの市場が立ち上がる

――ボーランドが日本市場をどのようにとらえているのかについてお聞かせください．

Wang　日本の市場をとてもエキサイティングなものと感じています．

我々が重要と考えている市場が，世界に3つあります．それは，米国，ドイツ，日本です．日本は現在，世界におけるボーランドの言語製品の売り上げの16パーセントから20パーセントを占めています．日本はすでに言語製品の巨大な市場なのです．

日本は現在も重要な市場であり，将来にはさらに重要な市場になると思います．近い将来，日本のプログラミング市場において，新しい技術への，つまりオブジェクト指向プログラミングとグラフィカルユーザーインターフェース，Windowsへの移行がおきるでしょう．そして，そのときこそオブジェクト指向開発ツールの市場が立ち上がると考えています．

3つのユーザーカテゴリに最適の開発ツールを提供する

――ボーランドの言語製品の位置付けについて聞きたいのですが．

Wang　言語製品のカスタマーとして，3つのタイプを考えています．

1つはプロフェッショナルなプログラマです．これはとても小さな市場です．中間に位置するのがアプリケーションデベロッパで，これはとても重要な市場です．最後がエンドユーザーです．彼等は必ずしもプログラマではありません．我々は，この3つのユーザーカテゴリにそれぞれの言語製品を一致させるようにしています．

プロフェッショナルデベロッパに対しては，Borland C++を用意しています．彼等は常に，プログラミングに関するすべてをコントロールすることを望んでいます．C++はとても複雑な言語で，初心者向けのものではありません．C++を学ぼうとしている人々のためには，Turbo C++を用意しています．しかし，Turbo C++を使ったからといって，C++が簡単な言語になるわけではありません．

よりシンプルな言語として用意しているのが，Turbo Pascalです．Turbo Pascalは，もちろんプロフェッショナルな用途にも使用できます．たとえば，AutodeskのCADソフトがTurbo Pascalで作成されていますし，ほかにも数多くのコマーシャルソフトが実際にTurbo Pascalで作られています．

Turbo Pascalは言語仕様が小さいので，C++よりも容易に学ぶことができます．

我々はPascalを拡張してオブジェクト指向プログラミングを実現するために，新しい4つのキーワード，object, virtual, constructor, destructorを加えました．この拡張で，オブジェクト指向プログラミングの基本機能である，カプセル化(encapsulation)，インヘリタンス(inheritance)，多態性(polymorphism)のすべてをPascalに実現したのです．Pascalは，オブジェクト指向プログラミングを学ぶ最も洗練された方法といえます．

最後に残った人々，エンドユーザーのために提供する開発ツールがObjectVisionです．ObjectVisionを使用することで，人々はビジュアルな操作だけで，プログラミングすることなしにアプリケーションを作成することが可能になります．さらに，C++やPascalでDLL (Dynamic - link library)を作成してObjectVisionの機能を拡張することもできます．

以上が，ボーランドにおける言語製品と市場の位置付けです．

最も重要なテクノロジーがアプリケーションフレームワークス

Wang　C++とPascalという2つのプログラミング言語に共通するもっとも重要な技術は，アプリケーションフレームワークスです．これは，なによりも重要なテクノロジーといえま

ワンラインNEWS
ソニーと日本電気は両者のWSで使う日本語アプリケーションソフト作成の際の規約を共同で開発したと発表した．それぞれ自社のWSのOSに採用する．

す．なぜなら，アプリケーションフレームワークスは，実際にプログラムの生産性・移植性を増加するからです．これによって，プログラマはより多くのプロダクツを作成できるようになります．アプリケーションフレームワークスまたはクラスライブラリを用いることで，DOS, Windows, OS/2上のハイレベルなプログラムを，より抽象化された形で構築することが可能になるのです．

将来，アプリケーションフレームワークスのカバーする範囲を拡大しようと考えています．たとえば，それはデータベースマネージメントであり，またインターフェースコンストラクションです．なぜなら，アプリケーションフレームワークスは，システムからも言語からも独立したものになるからです．プログラマは，DOS上でPascal言語からアプリケーションフレームワークスを使うこともできますし，OS/2上でC++からアプリケーションフレームワークスを使うこともできます．これらのコンセプトはまったく同じなのです．つまり，プログラマに高機能なオブジェクト指向ビルディングブロックを提供することです．プログラマは，アプリケーションフレームワークスを使用することで，大幅にプログラム作成時間を短縮することができるはずです．

プログラミングなしのアプリケーション開発を実現するビジュアルプログラミング

Wang　もうひとつの戦略が，ObjectVisionで実現されています．それは，ビジュアルプログラミングです．我々が提供しているビジュアルプログラミングは，まだ最初の世代のものですが，ビジュアルプログラミングにはまだまだ多くの可能性があります．将来的には，たとえばアプリケーションフレームワークスを部品として使うことで，スクリーン上でアプリケーションを絵を描くように描くだけでプログラミングがすべて終ってしまうようになるでしょう．

これらの製品を共に発展させながら，異なる市場のニーズに応えていくつもりです．ボーランドが供給する技術は，それぞれのユーザーに高い開発能力を与えることでしょう．

基本言語製品の改良とプログラマ支援ツールを拡充

――C++やPascal以外の言語製品を発売する予定はないのでしょうか．

Wang　現在新しい言語製品を用意してはいません．FORTRANはとても興味深い言語ですが，ボーランドはFORTRANを製品としては出していません．新しい言語製品を作る前にやらねばならないことが山のようにあるのです．FORTRANを製品としてリリースすることは可能だとは思いますが，アナウンスはしません．

今回のDevelopers Conferenceでは，いくつかの新しい製品についてのアナウンスを行いました．これらは，Briefプログラミングエディタ，バージョンコントロールシステムなど，複数のプログラマが共にアプリケーションの開発を行なう課程を支援する開発ツールです．このような開発支援ツールを提供すると同時に，基本的な言語製品，ビジュアルプログラミング，アプリケーションフレームワークスなどの新しい技術を改良していきたいと考えています．

優れたクラスライブラリによってプログラマは，大量にコードを書くことから解放される

――クラスライブラリの標準化を進めているOMG(Object Management Group) についてお聞かせください．

Wang　私は，OMGのボードディレクターを務めています．最近フロリダでミーティングがありましたが，ここでボーランドはクラスライブラリの標準に関する新しいスペシャルインタレストグループを作ることを提案しました．クラスライブラリの標準を決めることは，とても重要なことです．我々は，クラスライブラリの標準化がカスタマーに利益をもたらすと信じています．だからこそ，努力してObjectWindowsの仕様を公開し，OMGに提出しているのです．

マイクロソフトのMS-C/C++ 7.0に付属するMFC(Microsoft Foundation Classes)は，ObjectWindowsに取り入れるようなものではないと思います．なぜなら，MFCは単にC++版の SDK (Software Development Kit)に過ぎないからです．SDKを用いたプログラミングはとても良くないもので，誰もこのようなプログラミングをしたいとは思わないでしょう．ボーランドがC++で成功した基本的な理由の1つに，プログラマが皆SDKによるプログラミングを嫌っていたということがあります．

MFCはとても抽象化のレベルが低く，Windows APIに近いものです．これに対してObjectWindowsのコンセプトは，すべてのプログラミングをオブジェクト指向で行うというものです．1つ例をあげると，C言語で2200行になるマルチパッドのプログラムをMFCで記述すると1900行になり，ObjectWindowsを用いると156行になります．これはなんといってもデータ抽象によるところが大きいといえます．

我々の方法は決して完璧とはいえませんが，よりよい方法であることは確

かです．我々は，問題解決のための最も良い能力，最も良い考え方，最も良いデザイン方法をデベロッパに提供しています．その結果として，プログラマは大量のコードを書くことから解放されるのです．

オブジェクト指向への移行には2つの方法がある

――オブジェクト指向プログラミングの重要性とその有用性については日本においても認識されてきていると思うのですが，日本のプログラマの多くはとても保守的で，なかなか移行できずにいるのが現状です．

Wang　オブジェクト指向プログラミングへの移行には，2つの場合があります．

1つは，グラフィックスユーザーインターフェースに移行する場合です．このときが，オブジェクト指向プログラミングに移行するとてもよい機会であるということです．なぜなら，そうすれば山ほどコードを書かずにすむからです．

また，DOSの下では100以下のAPIファンクションコールしかありませんでしたが，Windows 3.1では750以上のAPIファンクションコールが存在しているのです．これはとんでもない数です！　オブジェクト指向プログラミングは，この複雑な問題を解決する唯一の解答なのです．

現在，Windows, OS/2，その他のOSの間でOS WARSといわれるものが起きています．これらのどのOSに対応したアプリケーションを作るかということは，デベロッパにとって大きな問題です．しかし，オブジェクト指向テクノロジーとオブジェクト指向開発ツールによって，これら複数のOSに対応したアプリケーションの開発が可能になるのです．

2つめは，グラフィックユーザーインターフェースに移行せず，DOSに留まっていた場合ですが，このときでもC++やTurbo Pascalを用いて少しずつオブジェクト指向プログラミングを試してみることができます．なぜなら，C++はCのスーパーセットなのでCの構文で書かれたプログラムとC++の構文で書かれたプログラムを混在しても問題ないからです．C++で作り上げたオブジェクトをプログラムの部品として，Cで書かれたプログラムに付加していくことができるのです．必要なだけオブジェクト指向テクノロジーを利用することで，ゆっくりとオブジェクト指向の機能を学んで行くことができるでしょう．

プログラマにはもっと多くの情報が必要だ

Wang　私は，日本においてオブジェクト指向についてより多くの情報がプログラマに与えられ，より多くの議論がなされるべきだと思います．アメリカにおいては，あらゆるコンピュータ雑誌，プログラミング専門誌がオブジェクト指向テクノロジーとプログラム開発効率についての情報を提供しています．

ボーランドは日本において，オブジェクト指向について解説し，コードを実際に示し，なぜオブジェクト指向プログラミングなのかを述べ，C++がよりよいCであることを示し，オブジェクト指向プログラミングが構造化プログラミングより優れていることを証明すべきだと考えています．

また，CプログラマがC++を学ぶ助けになるように，学習用のビデオテープの発売も開始しました．これらのビデオテープも徐々に翻訳されて，日本でも発売になるでしょう．

dBASE言語の拡張は最小限に，PALはよりオブジェクト指向ライクに拡張

――dBASE言語をオブジェクト指向化するということですが，どのような方法で行うのか，またPARADOXとの関係についてお聞かせください．

Wang　ボーランドは，データベースプロダクツに関する戦略を長年に渡って続けてきていますが，アシュトンテイト社を買収したことで，新たにdBASE製品の次世代バージョンであるdBASE for Windowsの開発を行っています．

dBASEの言語拡張は，できるかぎり最小限に押えるつもりです．なぜなら，dBASEは現在データベースアプリケーション作成言語の標準ですから，以前のバージョンとの互換性を保つことが最も重要になるからです．dBASE言語の最も大きな拡張部分は，Windowsに対応した部分になるでしょう．dBASE言語のシンタックスは，以前のものにより近いものになりますが，ユーザーインターフェースの部分ではオブジェクト指向テクノロジーの恩恵を受けられるようになります．

もう1つのデータベース製品であるPARADOXにおいても，現在PARADOX for Windowsを開発しています．PARADOXのアプリケーション開発言語であるPALは，Object PALという，よりオブジェクト指向プログラミング言語ライクなシンタックスを持った言語に拡張される予定です．

＊ここで，インタビューの最後にEugene Wang氏が教えてくれたちょっとした隠しコマンドを紹介しておこう．ボーランドのWindows対応言語製品のHelp | Aboutダイアログボックスを表示して，[Alt]+[i]を押下すると……，あとはやってみてのお楽しみ！

BORLAND INTERNATIONAL DEVELOPERS CONFERENCE

Visions

INTERACTIVE EDUCATION FOR THE '90s

ボーランド社が開催する年に1度の集い

さる4月12日〜15日の4日間にわたって，アメリカ，カリフォルニア州モントレーにて，Borland International主催のDevelopers Conferenceが開催された．アメリカをはじめ世界各国から5千人近いデベロッパを集めて行われたこのコンファレンスの模様をお伝えしたい．

Opening Reception/ Casino Night

Borland International Developers ConferenceはBorland Internationalが毎年行っているデベロッパ向け会議（お祭り？）だが，今年はBorlandの最大のライバルであるMicrosoft社がMicrosoft C（以後MS-C）のC++化に手間取っているスキに一気にBorland C++でシェアを伸ばしたことを受けてか，参加人数も昨年の数倍に増え，まるでBorlandの勝利集会のような様相を呈していた．実際，現在のアメリカにおけるBorland C++のシェアはMS-Cを大きく上回っている．

コンファレンス初日には，Opening Reception/Casino Nightとして盛大なパーティーが開かれた．カジノといっても本物のお金を使うわけではなく，Philippe Kahn氏の肖像が入ったBorland特製の紙幣を使って行うものだが，それでも皆しっかりと特製紙幣を握り締めて，転がるダイスや回るルーレットに目を輝かせていた．

適度にアルコールも入り，座が盛り上がってきたところで，Borlandのコンファレンスに欠かすことのできないPhilippe Kahn率いるご存じTurbo Jazz Bandの登場となる．今回は演奏だけではなく，Eugene Wang氏の「Borland C++」とかいう歌までが披露されてもうびっくり．歌詞は，「ルートなんとかを突っ走って，モントレーにやってきてBorland C++に出会ったよ……」とかいうものだったが，もしかするとそのうちBorlandの言語製品にオマケでEugene Wang氏の歌入りCDがついてくるかもしれない．それとも，Turbo Jazz Bandのライブビデオかな？

なにはともあれ，酒と博打とジャズの喧噪の中でコンファレンスの夜は更けていくのであった．

Opening Session

コンファレンス2日目から本格的にSessionおよびPresentationが始まった．オープニングは，もちろんBorland Internationalだ．まずは，Philippe Kahn氏からオブジェクト指向プログラミングの重要性・有用性，そしてデータベース製品における戦略が述べられた．将来的にInterbaseと呼ばれるデータベースエンジンを中心として，PARADOX, dBASE, スプレッドシートであるQuattroProなどから同一のデータベースに共通の操作でアクセスできるようになり，そのユーザーインターフェースも統一されるということである．また，ここでdBASE for WindowsとPARADOX forWindowsのβバージョンのデモンストレーションも行われた．dBASE for Windowsはすでにマルチメディアに対応しており，完成度の高さを伺わせていた．

続いてLanguage Business UnitのリーダーであるEugene Wang氏から，Microsoft C/C++ 7.0についての解説（？）が行われた．アメリカの比較広告が日本の比ではないこと



はよく知られているが，コンファレンスでのsessionもなかなか強烈である．Microsoftが広告の中で挙げているMS-C/C++ 7.0の利点をひとつひとつ検証し，結局これは利点にはならないといい切っていくのだ．しかも，スクリーンに示された広告文の上に「NOT!」の文字を赤くでかでかと張り付けて，さらに会場と声を合わせて「We are not!」と叫ぶという念の入れようである．日本でこんなことをしたら大問題だが，アメリカ人達はじつに楽しそうにしている．アメリカンユーモアってやつなんでしょう，きっと．

Eugene Wang氏は，いくつかBorlandの新製品を発表した．なかなか興味深い製品が含まれているので，ここで紹介したい．

- Borland C++ for OS/2
- ObjectVision for OS/2
- Turbo Pascal for Windows 1.5
- Brief Editor
- Version Control System

OS/2バージョンについては以前から出るといわれていたので驚きはしないが，BriefとVersion Control Systemには驚いた．この2つの製品は共にSolution Systems社から発売されているもので，Briefについては日本システム（株）が日本語版を発売している．どのような販売形態をとるのかはわからないが，統合開発環境を“売り”にしていたBorlandが独立したプログラミング専用エディタを発売するというのは青天の霹靂といっていいだろう．やはり，統合環境を好むプログラマが少ないということなのだろうか？ともかく，Windows版も出す予定があるということなので楽しみに待ちたい．


Turbo Jazz Band

Turbo Pascal for Windows 1.5（以後TPW1.5）は，現在のTPWのマイナーバージョンアップで，Turbo C++ for Windowsに付属しているObjectBrowserなどが追加されるということだ．

Presentation

3日間に行われたPresentationの大部分が，オブジェクト指向プログラミング・デザイン関連のものだった．使用される言語は，当然C++とPascalである．そのほかに，日本ではまだ発売されていないWindows上のビジュアルプログラミングツールであるObjectVisionが大きく取り上げられていた．ObjectVisionは，ぜひとも日本語化を進めて欲しい製品の1つである．ボーランド(株)の努力に期待したい．

Presentationはじつに楽しい雰囲気で進められ，講師の中には，自分が答に窮するような鋭い質問をした受講者にはマウスパッドをプレゼントすると宣言し，講義の最中にマウスパッドが会場を飛び交うという一幕もあった．

Presentationを受講してもっとも強く感じたのは，アメリカではすでにオブジェクト指向プログラミング，Windowsプログラミングが常識になっているということだ．自他共にオブジェクト指向テクノロジーのリーダーと認めるBorlandのコンファレンスだからということもあったのだろうが，受講者の発する質問のレベルの高さが強く印象に残った．

アメリカでは，日本では考えられないほどWindowsが普及しているが，これはPresentationに使われているマシンの価格が2000ドルもしないことを思えばなにも不思議なことではない．486DX，メインメモリ8MB，200MB HDDを内蔵し，VGA以上の解像度をもったマシンが，30万円しないのだ．この差が日本とアメリカのプログラミングの技術格差を生んでいるというのも，あながち間違ってはいないだろう．

ボーランド（株）によれば，このコンファレンスと同様のものを今秋日本で開催する予定だという．オブジェクト指向プログラミングの実力を知りたい，あるいはそれ以上のことを知りたいというプログラマにはまたとないチャンスである．ページ数の制限から，さわり程度のことしかお伝えできなかったが，アメリカに負けないすばらしいコンファレンスが日本で開かれることを願って，この記事の終りとしたい．

Special thanks to
Tamah Swenson
Rita Fox
Vance Christensen

（編集部・鈴木）

アキバ短信　特別版

香港電脳街レポート

香港に行ったついでに486マシンを買ってきた

DOS/Vの登場以来，IBM互換機の人気はうなぎのぼりである．なかでも，香港・台湾あたりでは，486マシンが日本での286マシン並みの価格で売られているという噂はかねがね耳にしていた．ところが，この4月，休暇で香港へ遊びにでかけようと考えていた編集子に，本誌「IC千夜一夜」の連載でおなじみの虎岩登氏から，486/33MHzのIBM互換機を買ってきて欲しいとの依頼があった．

ちょうど折りも折り，新聞社系の某パソコン雑誌が，編集部員をアメリカ，台湾，香港に派遣して，もっとも安い486マシンを見つけた者がそれを実際に購入してくるという企画を実施し（なんというゴージャスな企画であろうか！），香港がもっとも安かったという結論を出したところであった．果たして，本当に香港の486マシンは世界一安いのか？　旅行のついでに買って帰れるものか，実際に試してみることにした．

■3時間でマシンを組み立ててもらった

香港の秋葉原といわれる高登電脳中心は，香港の観光の中心地である尖沙咀から地下鉄で5つめの，深水埗にある．地下鉄の改札を出ると，地上への出口に向かう通路脇の壁に案内板があるので，表示に従って地上に出れば，目の前に大きな看板があるので，迷うことはないだろう．

香港の秋葉原が日本の秋葉原と趣が違うのは，電脳街自体が巨大な雑居ビルの中にすっぽりおさまっているというところだろう．このビルの1階には，スポーツ用品や化粧品などを売る店も入っているのだが，地下1階と2階はほとんどがコンピュータ関連のショップだ．書籍を売る店，ソフトを売る店，本体と周辺機器を扱う店など，店ごとにある程度傾向がはっきりわかれていて，ひとつひとつの店自体はそれほど大きくない．

さて，購入するマシンのスペックだが，虎岩氏からは以下のような要求が出されていた（本来は，予算とのかねあいでこれがダメだったらどうするか，思ったより安かった場合は，なにを追加するかなど実に細かい指定がなされていたことをつけ加えておく）．もっとも，編集子が細腕で日本まで持って帰ることを考慮して，CRTは購入しなくてよいとのことだった．

CPU 80486/33MHz
64K CASHE
メインメモリ　4MB
100MB　HDD
SVGA DISPLAY CARD
FDD/HDD CONTROLLER CARD
3.5" FDD×1，5"FDD×1
POWER SUPPLY
101キーボード
中くらいのタワーケース

パソコン本体を扱う店は2階に多い．雰囲気は秋葉原のラジオデパートなどと非常によく似ている．店の前に，そこで売っているマシンの代表的なスペックを書いたチラシが置いてあるので，それを見ながらパソコン本体を売っている店を何軒かチェックする．

次に，それぞれの店で，上のスペックにしたらいくらになるかを聞いてみる．ところで，お店の人にどうやって，この要求を伝えればよいのか？（当然ながら，香港の人は広東語と英語しかできない）．編集子は紙に書いて店の人に見せるという原始的な方法を採用した．これはかなり確実である．お店の人はその場でいくらになるか，見積金額を出してくれるので，相手のいった値段によっては，「それじゃぁキャッシュを256Kにしたらいくら？」とか，「ハードディスクを200MBに代えたら？」といった交渉も紙の上でOKだ（ただし，「486/50MHzでは？」と聞いたら，「It's very expensive」と一蹴されてしまった．店員は人を見るというのは万国共通である）．

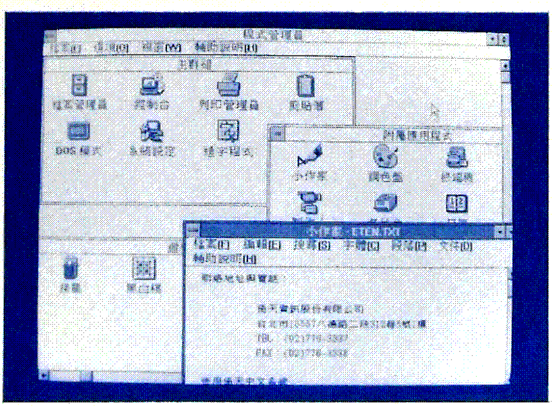
結局虎岩氏のスペックでは，1万香港ドルくらいということであった（1香港ドルは17～20円くらい）．ただし，ケ

ースはミニタワーになってしまうという．価格的には問題なく安いのだが，「日本まで航空便か船便で送ってほしい」というと，「壊れても保証できない」とか「送料がかかる」とかいって，なるべく持って帰るようにとすすめられ，ついにあきらめて手荷物にすることにして，価格の安かった店で購入することにした（結局あとになって，手荷物が正解であったことが判明する）．

さて，買う店と品物と支払方法が決まったところで安心はできない．ここから先が，日本で98を買うのとは大きく違うところなのだが，注文したパソコンはその場で引き渡してもらえるわけではない．店では客の注文したスペックにしたがって，マシンを組み立てるのである．実際に引き渡してもらえるのは半日から1日後ということらしいが，翌日の便で帰国する予定であったため，3時間で組み立ててくれるよう頼みこみ，その足でカートを買いにいく．慣れない香港で，パソコンの本体1台が載るような小さいカートを見つけるには，どうしようかと思案した結果，東急デパートのスーツケース売り場で92香港ドルの折り畳み式カートを発見．これを買って，また電脳中心へもどる．

■歩いている人の服装までアキバそっくり

約束の6時までにはまだ時間があったので，電脳中心の中を歩き回ってみることにする．パソコンを購入した店のある2階の一角は，比較的静かで，秋葉原でいえばちょうどラジオデパートの中のような感じだったが，地下に降りてみると，日本語の歌が流れている．どうやら工藤静香のようだった．これが森高千里だったら，秋葉原じゃないかと錯覚するほど電脳中心の内部の雰囲気は秋葉原にそっくりである．中を歩いている人たちの年齢層や服装までもよく似ているのがおかしい．



Windows 3.0 中文版

地下はソフトウェアや書籍を売る店がずらりと並んで，なかなかのにぎわいをみせている．なかでもびっくりしたのは，ソフトウェアの販売方法．噂には聞いていたが，コピーソフトが正々堂々と販売されている．ソフト名が書かれた紙が1枚と，5インチのフロッピーディスクがビニールの袋に入っているだけだ．その上からマジックインキでディスクの枚数と，値段がじか書きされている．「Windows最佳遊戯専輯(下)」は，Windowsのゲームらしいが，内訳はすべて漢字で書かれているので，想像するしかない．「水管狂想曲」（パイプドリーム？）「上海麻雀」（上海か？）はまだしも，「賭神之殿」だの「老虎機」というのはいったい何だろう……．

コピー品なのは，ゲームソフトばかりではない．ある店では，Windows3.0中文版をデモしていた．見れば後ろには紺と白のおなじみマイクロソフトのツートンカラーのパッケージが置いてある．聞けば日本円で約6,000円程度だというので，パッケージを指して「これが欲しい」というと，店員が奥から出してきたのは，「Windows易學易用専輯」という本が1冊と裸の5インチのフロッピーディスクが11枚！　2枚目のディスクにはごていねいにも「>SET-UP」と書かれていた．どうもこの国にはソフトの知的所有権などという概念は存在しないらしい．

たまに，やけに人が大勢集まっている店があるなぁと思ってのぞくと，たいていそれはファミコンのソフトを売る店であった．日本にもどってからあるファミコンソフト業界の人から聞いたところによれば，香港の生活水準からすると，ファミコンというのはかなり高価なおもちゃで，一般の人は店で遊ぶのがせいぜいなのだそうだ．ドラクエなどはやはり香港でもすごい人気で，ドラクエの画面に表示されるメッセージに広東語の対訳をつけた攻略本というのも存在するそうだ．残念ながらそのときは，そんなことは知るよしもなかった．

■店頭でDOS/Vを起動してみた

約束の6時に店へ引き取りにいく．15分ほど待ったが，それでもほぼ約束どおり，マシンは仕上がっていた．店の人が，サービスだといってソフトをインストールしてくれたが，その内容がすごい．LHArcで圧縮したインストール用のディスクを用意してあって，そこから次々と有名どこ

ワンラインNEWS
日立製作所は，5インチMOディスクの容量を現行の3倍にする次世代の記憶技術を開発した．記録部分の磁界を変化させる熱を制御し，高精度・高密度を可能にした．

ろのアプリケーションをハードディスクにコピーしていくのだ．Nortonか何かのベンチマークでCPUスピードやハードディスクのアクセススピード，RAM容量，ハードディスク容量などを表示させて，「ちゃんと入っています」とお客に見せるという手順になっているらしい．

日本から持っていったDOS/Vの起動用ディスクを入れて動作確認をしたいというと，3.5インチのドライブをAドライブにするからちょっと待てという．何もいわないと5インチのドライブをAドライブとして設定するようだ．現地の価格では5インチ2HDのフロッピーディスクが10枚で約50ドル，3.5インチだと倍の約100ドルくらいするせいだろうか．組みあがったマシンにDOS/Vのディスクを入れて起動すると，画面にはみごと日本語が表示された．画面に「ひらがな」が表示されると，見ていた店員さん一同から「おぉ」と声にならないどよめきがもれる．

ミニタワーケースをダンボールに詰め，キーボードを箱に入れてもらう．カートに積めば，楽々とまではいかないがひとりで十分持ち運べる大きさだ．クレジットカードで残金の支払をしたが，MASTER, VISAといった国際カードはかなりの店で使えるようだった．「THANK YOU」といって，店を出てしばらくすると，後ろから店員さんが追いかけてきて，VGAカードと本体の説明書を手渡してくれた．カートに載せたとはいえ，やはりパソコン1台は荷物になる．ホテルまではタクシーでもどった．

■虎岩氏はマシンをチェックした

さて，日本へのコンピュータの持ち込みだが，当初は機内へ手荷物として持って入るつもりだったのだが，ツアーの現地添乗員に「香港では座席の下に入らない大きさの手荷物は飛行機には持ち込めない」と頑強に拒否されて，おそるおそるカウンターで預けたが，どうやら外見には大した障害もなく日本に着いた．日本の税関では，海外での買い物が20万円を越える場合には申告することになっているが，パソコン本体以外ろくな買い物もしていなかったので，申告はしなかった．成田へ出迎えに来てくれた虎岩氏にマシンを引き渡して，香港486マシン購入の旅は一応終止符をうつ．

今回の買い物にかかった費用はつぎの通り．

マシン本体	10,000香港ドル
運搬用カート	92香港ドル
タクシー代	38香港ドル

1週間ほどして，虎岩氏からマシンの使用レポートと称するメールが届いた．まずはレポートの冒頭を一部ご紹介しよう．

> 最初に発見された不具合はだいたい発見順に以下のとおり
>
> 1．鍵が入っていない（後に発見）
> 2．ケースカバーのセルフタッピングビス斜め締め
> 3．ペリフェラル電源コネクタ嵌合不良
> 4．HDD取り付けネジ，片側省略
> 5．Trident SVGA cardのメモリのソケット挿入不良
> 6．基板取り付けセルフタッピングビス1本のみ，しかも締め込み不足
> 7．電源ユニット取付ネジゆるみ
> 8．表パネル取付ネジゆるみ
> 9．3.5インチFDD取付ネジ，片側省略
> 10．スピーカコネクタ外れ
> 11．Turbo LEDが点灯せず
> 12．電源コードは日本仕様でない（これはしかたないけど）

虎岩氏は，本体のカバーをはずして，中がきちんと組み上げられているかチェックを行ったのだそうだ．ご覧のとおり，よくぞこの状態で，無事に日本まで届いたものだ．逆にいえば，パソコンとは簡単には壊れるものではないという見方もできるかもしれない．ハードディスクなどは，ネジが片側しかとまっていなかったため，かなりグラグラしていたという．果たして，これが3時間で組み上げてもらったせいなのか，たまたま注文した店の仕事が雑だったのか，それとも香港では普通のことなのかはわからない．

また，本体のカバーをはずしたシャーシの裏に保証書が貼ってあったということだ．内容は，「購入時から12か月保証，ただしRAMとROMは90日間のみ．3年間は修理の際の技術料は不要．修理は店またはサービスセンターへの持ち込み修理のみ」とのことだった．

虎岩氏はその後，100時間の連続運転など，いくつかのテストを経て，現在はDOS/Vと日本語Windowsをインストールして快適に使用中であるとのことだ．

Book Review

■中村正三郎

C++らしいプログラミングの作法，骨法を伝える

アメリカではMS C/C++ 7.0が出荷され，日本でもBorland C++ 3.0の日本語版が出荷されようとしている．パソコン界でもCからC++への流れは，確固たるものになってきた．本誌の読者たるプログラマ諸氏もC++への移行中かと思う．その中で，C++の入門段階を終えて，中級あるいは上級を目指しているプログラマには，本書をお薦めしたい．

本書はC++の入門書ではない．したがって読み進むには，C++の知識とプログラミング経験が必要である．適宜，そこで用いられるC++の構文や概念の説明は出てくるのだが，それは知識と経験がある読者への簡単なおさらいと考えるべきもので，決してていねいな説明ではない．

では，本書が目指しているものは何か．それは，C++が提供する個々の構文や概念の先にあるものの提示である．

プログラミング言語を学ぶとき，文法と利用できるライブラリを学ぶのが通例だが，それだけでは，決していいプログラムは書けない．文法と言葉を知っているだけでは，決してよい小説は書けないのと一緒である．よい小説を書くためには，文法や言葉はもとより，小説作法，骨法といったものも身につけなければならない．

これはまた，プログラミングの世界でも同じである．文法やライブラリ関数の使い方を覚えた段階では，まだ初心者である．その言語が培ってきた，その言語なりの作法，骨法を学ばなければ，いわゆる「らしくないプログラム」になってしまう．「らしいプログラム」を書けるようになって，その言語を自家薬籠中のものとしたといえるのである．

本書は，C++らしいプログラミングスタイルとイディオムを数多く提示して詳しく解説している．イディオムとは，例えばCにおける while (*p++ = *q++); のような，その言語が浸透する中で育まれた慣用的コーディングのことである．先の言葉に置き換えれば，プログラミングスタイル=作法，イディオム=骨法となるだろう．これらを用いると，いかに強力な抽象化が行えるかを，本書は豊富な例を通じて示している．そして，オブジェクト指向による設計の話題にまで話は進む．評者は，読みながら興奮を覚えたほどだ．

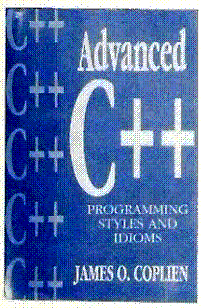
本書で明らかにされているC++のプログラミングスタイルとイディオムは，システム全体の設計，実装に大きな影響を及ぼすテクニックが多い．前述のCにおける例のような，枝葉末節的なちょっとしたプログラミングテクニックより，より深くプログラム全体の構造に関わるテクニックが目立つ．

これはC++のOOP機能から来る必然的結果ではあるが，言語によって培われる文化の違いを明らかにしている点で，非常に興味深い．CとC++のコミュニティは，ほとんどオーバーラップしているが，それでもこれほどの違いが出てくる．たしかにCからC++へと，言語仕様上の大きな飛躍があった．その飛躍が使う側にも起きていたことを示している．これは評者にとって新鮮な驚きだった．

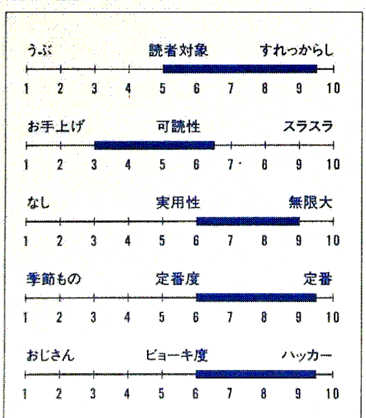
重要なのは，本書で示してあるプログラミングスタイルとイディオムが，

■Advanced C++ Programming Styles and Idioms

James O. Coplien 著
Addison-Wesley



■評価チャート



単なる思いつきではないことだ.

著者はベル研において，C＋＋が最初にできたときからC＋＋に関わり，豊富なプログラミング経験を持っている．その経験と知恵が盛り込まれている．また，ベル研内部外部を問わず，C＋＋コミュニティが，C＋＋誕生以来10年以上培ってきたプログラミングスタイルとイディオムが盛り込まれているのである.

C＋＋のパワーがどれほどのものか感じたい人，C＋＋プログラミングのヒントを得たい人，C＋＋の中級者，さらには上級者にも一読を薦めたい本である.

さまざまに利用されるMathematica，関連書籍が好調

古田島義和（神田・書泉グランデ）

Stephen Wolfram氏は，Mathematicaの開発者で，またWolfram Research，Incの創立者でもある人物です．今月は，このWolfram氏の作った「Mathematica」についてご紹介したいと思います．Mathematicaは，数学やそのほかの応用のための汎用ソフトウェアで，現在，多くの研究者，エンジニア，学生たちに使われています．Mathematicaの応用は量的な方法が使われる科学，技術，ビジネスなど幅広く，1988年6月にMathematica Version 1.0が発表されると，数学の技術分野ですばらしい評価を得ました．Macintosh対応版は1988年6月に出荷，続いて各種のワークステーション対応版が登場，1989年11月には386ベースのIBM PC互換機対応版が出荷され，1990年暮れには，さまざまなシステムで利用可能になりました.

Macintosh，CONVEX，DEC VAXおよびRISC，Hewlett-Packard/Apollo，IBM 386-PCおよび互換機（MS-DOSとMicrosoft Windows），IBM RISC，MIPS，NeXT，Silicon Graphics，Sony，Sun（およびSPARC互換機）と，主要な機種を網羅しています.

Mathematicaは，数学のみでなく，総合的なコンピュータのソフトウェアで，その用途は，質問を入力すると答を出してくれる数値計算と記号計算のできる電卓，関数やデータの視覚化システム，大小のプログラムを作ることができる高水準のプログラミング言語，テキスト，グラフィックス，アニメーション，サウンドを数学公式と組み合わせて対話的なドキュメントを作成するなどさまざまで，とくに数値的・記号的・グラフィックス的な処理をすべて統一的な方法で取り扱っています.

このようなMathematicaのさまざまな利用法を解説している図書が，各社から出版されています．Addison Wesley（以下AW）の「Mathematica 2/E」は，Version 2.0に対応，同社から訳本も出版されています．ほかにもAWの訳本がトッパンから，「Computational Recreations in Mathematica」（Mathematica：計算の愉しみ），「Mathematica for the Sciences」（Mathematica：理工系ツールとしての），「Programming in Mathematica」（Mathematica：プログラミング技法）として出版されています．また新刊予定では，AWの「Exploring Mathematics with Mathematica」，Prentice Hall（以下PH）の「Mathematica:A Practical Approach」，W.H.Freemaの「Mathematica in Action」が待たれます．最近の新刊では，Academic Pressの「Mathematica by jExample」，AWの「Beginner's Guide to Mathematica」，PHの「Self-Tutor for Computer Calculus Using Mathematica」（Burbulla.S.Dodson著）などが好調な売れ行きを見せています.

Mathematicaは，Professional Verisonは数十万円もしますが，Student Versionは3万数千円という手軽さで，Macintosh，Windows各対応版とも順調に売れています.

来月は，B.Meyer氏のObject-Oriented Sereiesの詳しい資料が入る予定ですのでご紹介したいと思います.

■和書＆洋書ベスト10

（書泉グランデ調べ：1992年4月1日〜4月30日）

	和書	出版社		洋書	出版社
1	MINIXを256倍使うための本	アスキー	1	New Hacker's Dictionary	M.I.T.
2	アラン・ケイ	アスキー	2	Modern Operating Systems	Prentice Hall
3	PC-9801スーパーテクニック	アスキー	3	Power Programming with RPC	O'Reilly
4	vi入門	アスキー	4	UNIX Network Programming	Prentice Hall
5	こんなもんいかがっすかぁ	アスキー	5	Programming Perl	O'Reilly
6	Inside X68000	ソフトバンク	6	SPARC Architecture Manual:Version 8	Prentice Hall
7	オブジェクト指向狂詩曲	技術評論社	7	Standard C Library	Prentice Hall
8	初めて使うGNU Emacs	啓学出版	8	Object-Oriented Modeling and Design	Prentice Hall
9	GNU Emacs Lisp manual	ビレッジセンター出版局	9	C++ Programming Langurage 2/E	Addison Wesley
10	PC-9801シリーズテクニカルデータブックBIOS編	アスキー	10	Learning GNU Emacs	O'Reilly

ワンラインNEWS
アイワは米コア・インターナショナルに資本参加した．出資金額は約500万ドルで，発行済み株式の26％を取得．両社は小型コンピュータ向け外部記憶装置を共同で開発する.

★ ★ ★佐々木哲の★ ★ ★

U.S. プロダクツウォッチ

★えっ！IBMが互換機を売るの？

とんでもない世の中になってきたものです．先頃IBMが低価格の台湾製の互換機を売ることになったという報道がありました．IBM PC以来，IBMアーキテクチャを打ち立て早10年以上，DOSとともに互換機市場を育て，IBM自身もその中で絶対の位置を確保してきました．

付加価値が大きく，また安価でもある各種互換機に押され，自身の売り上げは伸びてもシェアは失いつつあります．それにつれIBMがまず何かを開発しスタンダードを作って，互換機メーカーはそれに追随するだけという古きよき時代の構図はとっくに崩れています．

とくに近年，USでのすべての業界にわたる“体力が続く限りの安売り戦術”にまきこまれ，低価格機ではまったくといっていいほどIBM純正機は競争力を失っていました．

ここ数か月の報道では，とうとうIBMが台湾の互換機メーカーからPCを買い入れ，それをIBMブランドでメールオーダーを通じて売るということまでいわれていました．さすがに，これは実行に移されないようです．対Microsoftの喧嘩騒ぎでも知られているIBM副社長のJim Cannavinoは，今のところIBM自身がメールオーダーで互換機を売るところまではしないといっています．

IBMは，今まさにこの安価なマシンのマーケットでの戦術を学びつつあります．Northgateのような互換機メーカーに出資したり，互換機の重要な技術であるBIOSやマザーボード自体をOEMすることも始めるようです．もちろんこの背後には，IBMのもつコピーライトや特許があります．

混乱の続くUSのPCマーケット．その中でももっとも動きの激しい低価格機種マーケットはやはりしばらく目が離せません．

★XBase言語．ANSIで標準化へ

いくらNetWareが普及し，クライアントサーバアプリケーションの重要性が増してきても，今のところアプリケーション開発言語としては，絶対的にdBASEまたはその互換のものが標準です．

今では，Borlandに吸収されてしまったAshton-Tateですが，これが健在の時は，この言語仕様を巡って大きな争いがありました．ご存知のとおり，dBASEはとても広く使われているデータベース言語ですが，実際の業務に純正のdBASEパッケージを使う人は少数でした．アプリケーション開発者やVARは，実際にはその互換製品でコンパイラであるFoxBASEやClipperを使うのが常でした．理由は簡単．開発されたアプリケーションをdBASE本体なしで実行させることができますし，なによりも実行速度が数倍から数10倍高速だからです．

今では，FoxBASEのFox SoftwareはMicrosoftに買収されましたし，CilpperのNantucketもComputer Associatesに買収されましたが，これらをはじめdBASE互換言語はそれぞれ仕様を拡張してお互い互換性を失ってきていました．もともとのdBASEには，多次元配列もなかった状態なのでしょうがありませんが．

それに対して，Ashton-Tateはその末期にdBASE言語仕様そのものがAshton-Tateのものであり，他社による互換製品を許さないという姿勢でFoxBASE側やClipper側に訴訟を起こしたこともあります．それに対しユーザーおよび互換製品各社は，XBase言語としてdBASE互換言語を位置付け，その仕様を統一しようとしてきたのです．

最近やっとここ数年の努力が実り，ANSI XBase言語の提案がANSIにより採用されました．これにより，正式にANSIでその標準化について討議されます．秋のComdexあたりから最初のミーティングが持たれ，Borland，Fox Software，Microsoft，WordTech Systems，Recital，Emerald Bay Groupなどが参加します．

★最後の高速モデム規格V.fast

Rockwellなどの新しいチップセットの登場により，9,600bpsのV.32，14,400bpsのV.32bisモデムがとても安価になってきたのは皆さんもご存知のとおりです．低価格機では，FAX，VOICEサポート付で300ドル程度で購入できるようになってきました．もっとも，音声などの部分はまだソフトウェアが追いついていませんが．

CCITT V.32bis自体まだ制定されて2年ほどの規格ですが，すでに次の，そして最後の公衆回線用の高速モデムの規格となるであろう通称V.fastの制定が進んでいるようです．最終的に決まるには2年近くかかることになるかもしれません．

もっとも，先に見切り発車で製品は出てくるもので，Motorola Codex*1は一般公衆回線で24,000bpsを，また専用回線では115,200bpsまでの全二重の通信をサポートするモデムのシリーズを発表しました．もちろん，V.fastの規格は決まっていないため北米のユーザーは規格が決まり次第無料のアップグレードが受けられるというおまけ付きです．

＊1：326xFast series modem
3260Fast external model $1,395
Motorola Codex
Mansfield, Massachusetts
phone: (800) 334 9074.

★見たこともない！　デモ！

IBM PCの世界を見ていて楽しいのは，やはり“これは何だ！”，“何でこんなすごいことができるのか！”という驚きです．

このコラムがUSウォッチとなっているように，新しいものを見るのにアメリカばかりに目が向きがちです．ところがところが，ヨーロッパも面白い．

最近，Vector demoとかCRONOLOGIAというような驚くべきソフトウェア群と出会う機会がありました．これらは，オランダやデンマーク，ノルウェー，スウェーデンといった北欧各国で書かれたもので，VGAの載ったマシンとSound Blasterなどの音源ボードで動作します．用途は……デモプログラムなのです．あるいは人を驚かせ，PCの前に縛りつけあげくは次の日会社を休ませてしまうようなもの．それ自体は何をするわけでもないのですが，画面にメッセージとともにIBM PCでは見たこともない，とても不可能と思われていたような画像を表示し続け，また，Sound Blasterの機能をフルに生かしたバックグラウンド音楽を聞かせてくれます．PCの画面というよりは，よくできたMTVを見ているような気がしてきます．

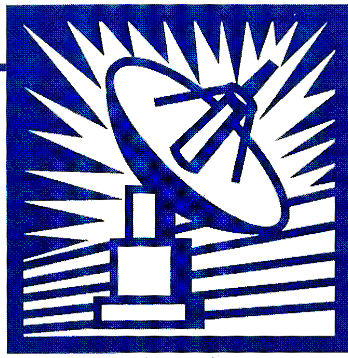
これらのデモプログラムは，北欧各地にあるPCユーザーグループの自己主張のデモなのです．何かの機会につけては自分たちのプログラミング，チームワークの威力をデモする．ユーザーグループミーティングやBBSを通じて彼らの成果を交換しあう．ヨーロッパは，北米とは異なりAmigaやCommodoreなど異種の文化が盛んです．IBM PCとはまた異なった文化．その結果，これらのデモはとても新鮮に感じられ，また私たちのUS中心の考え方に疑問を投げかけてくれます．VGAにしても色使いから違います．音楽にいたってはSound BlasterといえばFM音源として，いわゆるゲームの音しか知らなかったのが，デジタイズされ聴きやすくミキシングされ，しかもデータは少なく効率よく使われています．

いま日本では，486マシンばかり盛んに売れています．ところが彼らのハードウェア構成を見ると実に貧弱です．それが故にプログラミングでカバーしています．

これらのデモソフト群は，日本ではNifty-ServeのFTRIALフォーラムで入手可能です．GO FTRIALの後，ライブラリの5番を見てください．

VECDEM.ZIP（206248バイト）
　Vector demo．テクノサウンドのBGMとともに3Dの物体が飛び交う．［オランダ］
CRONOLOG.ZIP（473622バイト）
　CRONOLOGIAデモ．4部作からなる大作．3D，MTVなどなど．［スウェーデン］
FISHTRO.ZIP（240591バイト）
　水をテーマにしたもの．［フィンランド］
VICKY.ZIP（244071バイト）
　物静かな3D物体のデモ．［デンマーク］

Illustration : NOBUKI FUKUHARA



西川和久の

希望的観測

ビジネスショー開催される!!
今年の面白いネタは……

1 SHARP書院パソコン ワープロ他メーカーの動向は？

今回なんといってもトップネタはこの「書院パソコン」である．従来のSHARP書院をそのままに，OADG DOS/V仕様を盛り込んだパーソナルワープロなのだ．カタログのキャッチフレーズも“何故いままでなかったのだろう．「書院パソコン」，ついに誕生．”と書かれている．これまでなかった理由は簡単であり，「メーカーサイドから見ると２度，３度美味しい市場を一本化したくなかった」からである．即ち，まず下位のパーソナルワープロを購入した人がしばらく使い，能力不足から上位のパーソナルワープロに買い換える．次はワープロ以外のこともしたくなり，ノートパソコンを購入，そしてデスクトップへ……商売上手なメーカーがこんな美味しいことをわざわざ自ら崩壊する理由などどこにもないのだ．今回の書院パソコンも，いろいろな理由から社内ではかなりモメタのではないだろうか??

パソコンとして考えると386SX/16MHzまたは20Mz，メモリ２MB～６MB，ハードディスク０～80MBなど，平均的なスペックである．気になる価格は330,000円～590,000円までとちょっと高いような気もするが，400dpiの熱転写プリンタ，アウトライン・フォント，ワープロ/通信ソフトなどが標準で付属していると思えば妥当なところであろうか!?　外観も従来の書院とほとんど同じであり，何も説明書きがなければ見過ごしてしまいそうになる……それほどワープロ・ワープロしているのだ．このようなマシンが家電ルートでワープロ＋αとして販売されれば，DOS/Vユーザー層も一気に広がるであろう．筆者も１台自宅に欲しいマシンである．(^^;

このほかにもビジネス用のツールを多く展示しており，全体的にちょっと価格が高めなのが気になる点であるが，ペン入力で電子手帳コンパチの“ハイパー電子マネージメント手帳”や“名刺読取機”など，パーソナルOAを主体としていた．

残念ながら今回のビジネスショーでは，NEC，松下，富士通，東芝など主要パーソナルワープロメーカーから，このようなマシンは発表されなかったが，今後の動向が非常に面白くなりそうだ．

2 COMPAQのノート型CONTURAシリーズ なんとキャラベルからもDOS/Vマシン

海外同様，日本においても企業狙いのCOMPAQであるが，このノートパソコンはちょっと違うようだ．４モデル発表され，かなり頑張った価格設定となっている．各モデルの数字はハードディスクの容量を表わし，COMPAQ CONTURA 3/20　(CPU:386SL 20MHz，RAM:2MB) model 40 298,000円/model 84 393,000円，COMPAQ CONTURA 3/25　(CPU:386SL 25MHz，cache 64KB，RAM:4MB) model 60 398,000円/model 120 498,000円なのだ．重量2.8Kgでバッテリー駆動３

時間と平均的なスペックであるが，CPUやハードディスク容量などを考えてもこれはかなり安い!!　パーソナルでも十分購入可能な範囲である．片面のカタログ1枚なのでこれ以上詳しいことはわからないが，他メーカーにも十分太刀打ちできるこのマシン，国内に与える影響は大きいであろう．

また，486DX2/66MHzのマシンやカラーノートなど新製品も展示され，なかなか気合の入っているブースであった．

あのハードディスクで有名なキャラベルがDOS/Vマシンを発表した．486SX/20MHz，486DX/33MHz，486DX/50MHz（EISA）とミニタワー型モデルのラインナップを引っ提げて登場である．これには正直驚いてしまった．確かにPC互換機は各パーツごとに組み合わせれば，簡単にオリジナルマシンができあがってしまうが，まさかハードディスク・メーカーまで参入とは……．当然，今後ますますこの傾向が強くなことが予想できるため「面白い時代になったな!!」というのが実感である．

3 Canonの高解像度FLCディスプレイで1280×1024のWindowsをデモ

今回，筆者がビジネスショーを見て一番欲しいと思ったのが，この液晶ディスプレイである．これまでカラー/モノクロに関らずパネルタイプ・ディスプレイでパソコンに接続可能な解像度は最大640×480であった．しかしこのFLCディスプレイはカラー/モノクロ共に15インチで解像度1280×1024なのだ!!　米国版のWindowsを動かしデモしていたが，反応やコントラストも十分である．値段を聞いたところ，参考出典のために未定であったが，モノクロタイプなら購入可能な範囲ではないだろうか??

残念ながら，2つともATバスのカードにビデオチップと液晶駆動回路を搭載しているため，汎用的に使うことはできないが，日本語Windowsでぜひとも使ってみたいものである．もし，日本語Windows用のディスプレイドライバがなくてもDDDがあれば安心である．:-)

4 ハードな季節だあー疲れた……

朝の早くから夕方まで，記事ネタを探して会場内をうろうろするが，今年も数多く見るところがあり足が棒になってしまった．また，この時期は各地からいろいろな人が集まるため廃人宴会も多発し，外出が多くなるためにコールバックの山，暴飲暴食，寝不足，ホコリと話し過ぎによるのどあれ……　(^^; 布団が恋しい季節である．



コンピュータの法律

コンピュータ取引の法律問題

熊谷信太郎（弁護士）

■コンピュータ取引――ハード

コンピュータ取引の法律問題は，コンピュータのハードおよびソフトの双方について，問題となります．

ハードについての契約は，ハードの売買や賃貸借の関係と，保守管理の関係に分けることができます．

このうち，ハードの売買契約は，従来，あまり多くは使われてきませんでした．というのは，一度買ってしまうと，新しいハードが出たときに，そちらに乗り換えるのが難しいことや，売買代金が高額のために，賃貸借の形式をとる例が多かったためです．

ハードの賃貸借契約には，リース方式とレンタル方式があります．両者は似ていますが，契約の途中で解約することが自由かどうかがちがいます．

リース方式の場合には，契約途中の解約は制限されることが多いかわり，リース料は割安です．レンタル方式の場合には，契約途中の解約ができることが多い反面，リースに比べて料金は割高に設定されています．

ハードを借りる場合には，このような両者の性格のちがいを認識してうまく使い分けていくのが得策です．たとえば，高性能なハードで，当分の間バージョンアップを考えなくてもよいという場合には，長期間のリース方式を選んだほうがよいし，逆に短期間だけ借りたいという場合や，将来より性能のよいハードが出たらすぐに切り換えたいというような場合にはレンタル方式が優れています．

もっとも，リース方式といっても単一ではなく，さまざまなものがあります．リース方式は，大別すればメーカーが自ら行うもの（オペレーティング・リース）とリース会社が行う（ファイナンス・リース）の2種類に分けられます．

両者はいろんな点で異なりますが，その特徴を簡単にいえば，オペレーティング・リースは，メーカーがハードを売る代わりに貸す契約で，比較的短い期間が設定されることが多く，ハードの価格を一定の月数で割った額が毎月のリース料になります．また，リース期間中の任意の解約も認められるのが一般です．

これに対し，ファイナンス・リースは，契約の形式としては賃貸借ですが，その実態は，リース会社がユーザーに対し融資をすることを目的とする契約です．つまり，リース会社はハードの価格にリース期間中の金利や費用を上乗せした金額をもとに，毎月のリース料を計算します．そして，リース期間はハードの耐用年数と同じ期間設定されるのが一般であり，その間の中途解約は認められないのが普通です．

■コンピュータ取引――ソフト

これに対しソフトの契約は，ソフト開発契約とソフトの使用許諾契約，ソフト譲渡契約に大別することができます．

ソフト開発契約は，自らソフトの開発を請け負う内容の契約と，他人の行う開発行為を援助する契約とがあります．

ソフト開発の請負契約は，請け負った側がソフトの開発について責任を負担しますので，料金は高額になることが多く，またその場合，プログラム著作権の帰属の問題が生じてきます．プログラム著作権は，開発を請け負った

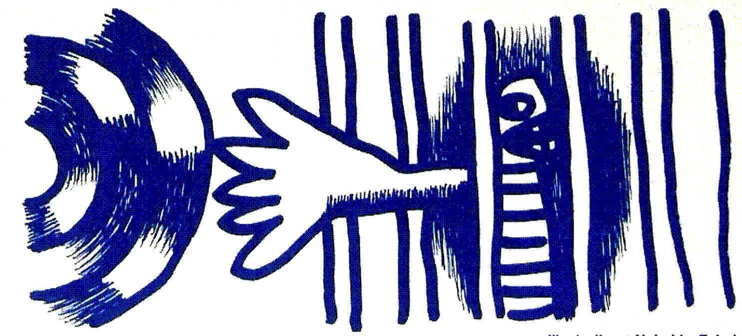


Illustration : Nobuki Fukuhara

者にまず帰属したうえ，代価(委託料)の支払いと引き換えに委託者に移転すると考えられますが，詳しいことは，本誌1991年12月号から1992年４月号をご参照ください．

ソフトの開発援助契約は，委託者が行うソフトと開発を受任者が援助するもので，開発請負契約と異なり，受任者側はソフトの開発そのものについての最終的責任は負担しません．当然，料金も開発請負契約にくらべて割安に設定されています．

これに対しソフト使用許諾契約は，ソフト所有者が有償でソフトの非独占的使用権をユーザーに許諾するものであり，同一のソフトにつき多数の使用許諾が可能になります．これによりソフト所有者は，ソフト開発費，またはソフト取得費を回収することが可能となり，一方，ユーザーはソフト開発の費用やリスクを負うことなく，ソフトの利用をすることができ，しかも使用料は税法上経費として処理することができるので，この契約は非常に多く利用されています．

これに対しソフト譲渡契約は，ソフトの所有権をほかに譲渡する契約ですが，譲渡後はソフト上の権利は売主から買主に移転してしまい，売主は権利を失うので，複数の利用権設定によって資金を回収できる使用許諾契約と異なり，実際にはあまり使われていません．

アキバ短信

ラオックスTHE COMPUTER GAME館

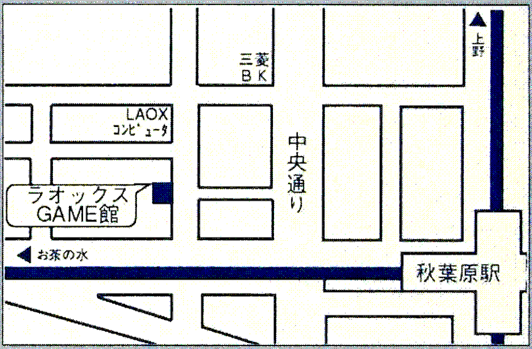
ついに登場したCOMPUTER GAMEの専門店

「バブルがはじけた」という台詞は，もはや時候の挨拶としてすっかり定着した感がある．ひところは，パソコンの売り上げも頭打ち，そのあおりでソフトウェアも売れず，パソコン業界も冬の時代を迎えるのかといった観測もあったのだが，実際のところはそれほどでもないらしい．NECの新製品発表を前に，一時的にユーザーが買い控えをしたのが少々大げさに伝わっただけだという説すらある．ともかく，秋葉原界隈は相変わらず活気がある．

さる４月の29日，いまや秋葉原随一のコンピュータのデパートとして，誰もが一目おくところとなった感のあるラオックス THE COMPUTER館の裏手に，THE COMPUTER GAME館がオープンした．スーパーファミコンをはじめとするゲーム専用機から，PC-9800シリーズ，FM TOWNS，X68000，MSXなどの国産のパソコン，IBM，Macintoshにいたるまで，ありとあらゆる機種に対応したゲームソフトを１階から５階までのフロアにずらり取り揃えた，まさにコンピュータゲームの専門ショップである．

フロアの構成は，１階がファミコン&スーパーファミコンのハードとソフト，２階がメガドライブ，PCエンジンなどのハード，ソフトと攻略本，ゲームミュージックのCDやビデオなどのコーナー．３階はNEC PC-9800シリーズ，４階はFM TOWNSとX68000，MSX，５階はIBM PCとMacintoshのそれぞれハードウェアとゲームソフトという機種別になっている．万引き対策ということなのか，

営業時間　10:00〜19:30（ただし日曜は19:00まで）
TEL　03-3251-3100　年中無休



１，２階はゲームソフトのパッケージの写真のパネルが，価格とともに掲示されているだけで，実際のソフトウェアのパッケージは，購入するまで手にすることはできないが，３階以上はすべてパッケージを手にとって見ることができる．また，３階には，ブラザーのソフトウェアベンダー武尊が置かれている．

THE COMPUTER GAME館は，４月初旬までラオックスの楽器館だった所だ．THE COMPUTER館との間には，先ごろソフマップがスーパーファミコン，ファミコン，ゲームボーイの中古ソフトを専門に扱うソフマップ11号館をオープンしたこともあって，この界隈は中央通りぞいの通称「３軒並び」(ただし，現在では３軒はおろか５〜６軒のゲームソフト専門店が軒を連ねている)と並んで，コンピュータゲームフリークには，見逃せないエリアとなったわけだ．

完成間近と噂されているスーパーファミコン対応の「ドラゴンクエストV」が発売されるころには，ここにも長蛇の列ができるのだろうか．

ワンラインNEWS
日本電気は仏ブルグループとICカード用半導体チップの特許契約を結んだ．ブルと特許契約を結んだ国内半導体メーカーでは４番目．

●プログラマの仕事場 | 16

PW Programmer's Working Space

インテルジャパン株式会社
PLV動画圧縮サービスセンター

マルチメディアタイトルの開発をサポートするデジタル画像処理の拠点

マルチメディアでパーソナルコンピュータの表現を広げていこうという方向はすでに定められているようだ．しかし，現実の問題としてマルチメディアを扱うことのできるハードウェアはまだ普及しているとはいえないし，ソフトウェアの数も限られている．今回は，少し視点を変えてマルチメディアの基礎技術とそれをサポートする体制づくりについてふれてみよう．

マルチメディアを実現するには画像データの圧縮技術が必要

マルチメディアにおいてまず解決しなければならないのは画像データの取り扱いである．画像ファイルはサイズが大きい．たとえば，Windowsの壁紙であるbmpファイルを見てみても，ディスプレイの全画面の大きさのファイルは百数十KBから数百KBにもなる．画像を扱うにはこのサイズが問題だ．大容量の記憶媒体が必要となるし，たとえ媒体の問題がクリアされても，ファイルサイズが大きいと表示に要する時間がネックになる．1画面を表示するのに数秒もかかるようでは，1秒間に30コマ程度を要する動画を取り扱うことなど無理というものだ．

マルチメディアの記憶媒体としては，CD-ROMが標準になると見られる．フルカラー（1677万色）のデータで動画の1コマが720KBだとすると，CD-ROMの650MBの容量をもってしても，30秒程度の動画しか収録することができない．また，1秒間のデータ量は21.6MBにもなるのに，CD-ROMのデータ転送速度は1秒間に150MB程度だから，動画の再生は無理だということになる．

データサイズの問題を解決するには，データを圧縮して記録すればいい．インテルの提供しているDVI®技術（デジタル・ビデオ・インタラクティブ）は，そうした必要性から登場してきた技術だ．この技術は，圧縮・伸長を行うチップセットを搭載したボードなどで実現される．容量的に収録が可能であればCD-ROMでもハードディスクでもよく，メディアを限定するものではない．

この技術によって，対話型のインターフェースをもち動画と音声の入った新しいタイプのソフトウェアタイトルが実現できる．また，データがデジタルであるため，1ピクセルごとの制御が可能となり，現在はビデオスタジオでしかできないような画像編集を机上で行うこともできるようになる．

160分の1にまで圧縮すればCD-ROMに72分の動画が入る

DVIでは，画像データを160分の1にまで圧縮できる．そうすると，CD-ROM 1枚に72分のカラー動画を収録することができる．DVIは，専用のチ

ワンラインNEWS
インテルジャパンは8Mビットのフラッシュメモリを発売．8Mビットが製品化されたフラッシュメモリでは最大の容量で，これを組み込んだメモリカードも順次発売する．

ップセットを搭載したボードによって実現されている．たとえば，IBM PC互換機にDVIボードを挿せば，DVIによるマルチメディアアプリケーションの実行が可能になる．媒体がCD-ROMならそのドライブも必要になる．また，開発用のシステムと再生用のシステムは分けて考えられている．開発用には，リアルタイム圧縮が可能なボードとコントロール画面用のディスプレイが必要だ．再生機は，パソコンの形をしている必要はなく，本格的な普及はDVIプレーヤーによって担われるはずだ．プレーヤーの試作機も何機種かがすでに発表されている．

特筆すべきなのは，圧縮・伸長を行うプロセッサがマイクロコードによって制御できることだ．これによって，新しい画像圧縮の規格が制定されたとしても，それをマイクロコードで追加することができる．また，アプリケーション開発者はデジタル画像処理のロジックなどをマイクロコードによって書くことができる．

DVIの開発用システムでは，パソコン上でリアルタイムな画像の圧縮・伸長が可能だが，実際に出荷する段階ではより画質の優れたPLV (プロダクションレベル・ビデオ) と呼ばれる方式がとられる．この方式では，デジタル画像をスーパーコンピュータによって，解析したうえで圧縮処理する．今回紹介するPLV動画圧縮サービスセンターは，そのPLV方式による動画圧縮をサポートする．



スーパーミニコンでデータを解析・圧縮

インテルジャパンの筑波本社内にある同センターは，日本に2か所あるDVI動画圧縮サービスセンターのひとつだ．もう1か所は，日本コロンビアにあり，DVIソフトウェアの開発者にとって，このセンターは貴重な存在である．

実際に動画データの圧縮を行うシステムは，空調のきいたマシンルームに設置されている．860プロセッサを並列したiPSC (インテル・パラレル・サイエンティフィック・コンピュータ) が稼働しているため，パソコンのように室温というわけにはいかない．ほかにも，コントロール用のワークステーション，画像確認用のモニターディスプレイ，ビデオ機材が並ぶ．

圧縮サービスは，発注者が処理の仕様を指定したリクエストフォームとビデオテープをインテルに渡し，仕様に沿ったデータ圧縮をしてからデータを返却する流れで行われる．発注者は原画像データを1インチの業務用ビデオテープかD2テープに納めインテルに渡し，インテルでは圧縮データをストリーマテープに納めて返却する．

オフライン圧縮の利点として，インテルジャパンは次の3点をあげている．

- **非対照圧縮アルゴリズムによって画質を改善できる．**
- **非対照圧縮アルゴリズムは，原画像の特性を解析し，動き補正などの処理を行うため，高画質・高圧縮率で動画を処理できる．**

About Company

インテルジャパン株式会社．86系CPUのメーカーとして知られたインテル社の日本法人．エンドユーザーにも知名度を高めようと，テレビ，新聞広告，などで「Intel in it」キャンペーンを実施中．

Location

東京本社と筑波本社の2本社方式．筑波本社は，つくば市の科学万博記念公園の近くにあり，PLV動画圧縮サービスセンターはそこにある．周辺には，製薬会社や電子部品メーカーが多く，道路の両側の広い敷地に低層の建物が並ぶ風景はアメリカ的．東京本社は，東京駅に近い新丸の内ビルにある．

Products

・86系CPU

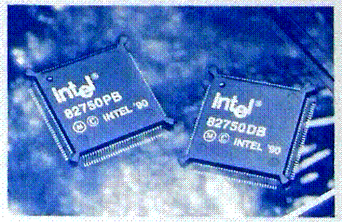
世界中でパソコンに搭載されている86系CPUはあまりに有名．8086, 286, 386, 486と世代を経てきて，現在では386以降の32ビットCPUを搭載したパソコンが主流になっている．しかし，ハードウェアの進歩にソフトウェアが追いつかない状態で，パソコンの標準OSはまだ16ビットベースのMS-DOSだ．32ビットCPUの機能を活用したOSとして，OS/2 2.0, WindowsNTなどが控えているが，普及には時間がかかるだろう．

新しいCPUの開発は，これまでのチップとの上位互換性を確保しながら進められており，現在は，次世代とその次の世代のプロセッサの開発チームがすでに活動しているという．

・DVIボード

「アクションメディア®II」という製品名で，DVIソフト再生用のデリバリボードとデータ取り込み用のキャプチャモジュールが供給されている．

DVIビデオプロセッサ82750



・DVI
DVI技術は，複雑な経過をたどって発展してきた．1983年，RCAのデビッド・サーノフ研究所において研究が始まった．それが，GEに買い取られ，さらに1988年にはインテル社に技術と研究員が引き継がれた．

・圧縮時にデータ処理を行っているため，伸長時には能力の小さいプロセッサでも処理することができる．

高速なスーパーコンピュータにより，データを解析し圧縮することで，CD-ROMなどに納められたDVIソフトは，パソコン，DVIプレーヤーにおいて優れた画質で再生できることになる．

画像の動きの差分と色情報を4ピクセルおきに記録

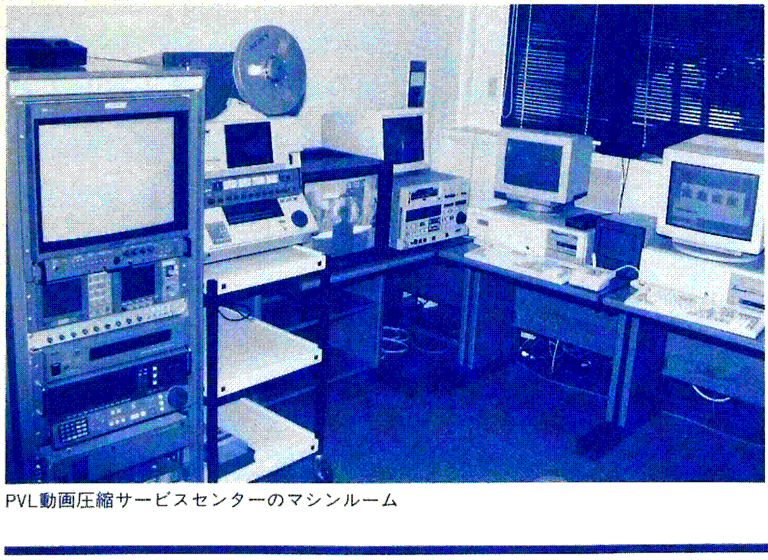
どうして160分の1まで圧縮が可能なのかについて，もうすこし説明しよう．

圧縮されたデータは，リファレンスフレームとデルタフレームによって構成される．リファレンスフレームとは，動画の1コマ分をそのままデジタル化したデータであり，デルタフレームとは，前のフレームとの差分を記録したデータだ．差分だけを記録したフレームを作ることによって，データ量を少なくすることができる．

リファレンスフレームとデルタフレームの割合は，データ処理を行うときにコントロールできる．リファレンスフレームが多いほど画像は良くなるが，反対に圧縮効果は低くなることになる．また，動画の内容によっても圧縮率は変化する．頻繁にカメラアングルを変えたり，動きの激しい動画は圧縮率がよくない．インテルでは，120のデルタフレームに対して1コマのリファレンスフレームをいれることを推奨している．

さらに，色情報に対してもデータ量を減らすための処理が行われる．縦，横とも4ピクセルおきにだけ色情報を持つように処理される．これによって，色情報は16分の1にまで抑えることができる．省かれた色情報は，再生する時に補正処理される．

こうした処理を行ったデータに対して，ファイル圧縮アルゴリズムによるプロセスが加わり，最高160分の1というデータ圧縮が可能になる．


PVL動画圧縮サービスセンターのマシンルーム

リアルタイムシステムではISDN回線でビデオ会議ができる

同センターは，PLVの圧縮サービスを行っているが，もうひとつの方式として，RTV（リアルタイム・ビデオ）がある．これは，DVI開発用のパソコンシステムでリアルタイムの動画圧縮・伸長ができるものだ．ソフトウェアタイトルを制作するときにあらすじを作ったり，プレセンテーションに使ったりという用途があるが，通信で動画を送ることにも利用できる．通信回線は，1秒間に伝送できるデータ量に限りがある．現在のテレビ電話が静止画なのはデータ量の制限によるものだ．そこで，データを圧縮して送れば動画も可能ということになる．

今年のインテル社株主総会において，米国の会場と筑波のインテルジャパンを結んでビデオ会議のデモンストレーションが行われた．回線は，INS64の2回線で1秒あたり256Kビット（4B：64K×4）のデータ伝送量を確保し，128×120サイズの画像を7フレーム／秒で送受信した．1秒あたり7フレームだと動画としてはかなりコマ落ちした印象になるが，相手の表情や動きがわかるため臨場感のあるデモンストレーションとなったようだ．

注目すべきなのは，日米間でビデオ会議が行われたということではない．TV中継回線があればもっと画質のいいテレビ会議が可能だろう．PCとISDN回線で，実現されたことに意味がある．それは，パーソナルレベルで動画と音によるコミュニケーションが可能であることを意味しているからだ．パーソナルレベルでアクセスできる情報が大きく広がることになる．

ワンラインNEWS
名古屋大学工学部の毛利教授とユニチカの中央研究所は，新しい動作原理を持つ磁気ヘッド用素子を開発．地場を直接検出するのでディスクが静止していても情報が読める．

●イベント

■6月24日(水)～6月27日(土)
CIMジャパン'92
製造業のための情報戦略・統合生産システムの展示会. FNE '92 (Fais Networking Event) も同時開催され, 工場自動化相互接続システム (FAIS) のデモンストレーションが行われる.
会場：日本コンベンションセンター (幕張メッセ)
入場料：5,000円
問い合わせ先：リードエグジビションジャパン(株)
TEL 03-3349-8501

■7月2日(木)～7月4日(土)
ファインプロセス・テクノロジー・ジャパン'92
液晶を中心としたフラットパネルディスプレイの製造装置と材料の専門展示会.
会場：国際見本市会場 (東京・晴海)
入場料：3,000円
問い合わせ先：リードエグジビションジャパン(株)
TEL 03-3349-8502

■7月7日(火)～7月9日(木)
PC WINDOWS World/Expo Tokyo
Windows関連製品の展示会と基調講演, フューチャービジョンなどをテーマにしたセミナーのほか, ワークショップなど約52のセッションが行われる.
会場：日本コンベンションセンター (幕張メッセ)
入場料：【コンファレンス・展示会3日間共通】30,000円 (前売り) 35,000円 (当日), 【展示会3日間共通】2,000円 (前売り), 1,500円 (当日1日券)
問い合わせ先：コンファレンス事務局
TEL 03-3359-8318

■7月7日(火)～7月9日(木)
AI'92
AI(人工知能)関連のアプリケーション, 開発支援システムなどを展示.
VOICE SYSTEMS WORLDWIDE '92 JAPAN
ボイスメール, 音声認識装置などの音声システム関連製品を展示.
リスク&セキュリティ・マネジメント'92
高度情報ネットワーク社会における安全性, 信頼性を確保するための製品を展示.
会場：東京流通センター (東京・平和島)
入場料：1,000円 (3展共通), 800円 (20名以上団体)
問い合わせ先：日本経済新聞社事業局総合事業部
TEL 03-3243-9083

■7月13日(月)～7月15日(水)
光メモリシンポジウム'92
光ディスク, 光カード, 光メモリなどの専門家が一堂に会し, とくに応用システムに重点を置いたセッションを行う.
会場：パシフィコ横浜 (横浜市・西区)
参加費：【6月25日以前申込】25,000円 (会員), 30,000円 (非会員), 5,000円 (学生)
【6月26日以降申込】28,000円 (会員), 33,000円 (非会員), 5,000円 (学生)
問い合わせ先：(財) 日本学会事務センター
光メモリシンポジウム'92事務局　TEL 03-5814-5800

■7月13日(月)～7月16日(木)
第5回国際マイクロプロセスコンファレンス
「リソグラフィ技術および装置」,「エッチング, 堆積技術」などをテーマに応用物理学会が主催するコンファレンス.
会場：かながわサイエンスパーク (川崎市・高津区)
参加費：【4日間】40,000円 (一般・6月30日以前申込), 45,000円(一般・7月1日以降申込), 10,000円(学生)
【1日券】20,000円 (一般・会場のみで受付)
問い合わせ先：(財)日本学会事務センター
マイクロプロセスコンファレンス係　TEL 03-5814-5800

■7月14日(火)～7月17日(金)
インターオプト'92
最新の光産業関連製品, 技術の展示会, セミナーも開催される.
会場：日本コンベンションセンター (幕張メッセ)
入場料：無料
問い合わせ先：(財)光産業技術振興協会インターオプト事務局　TEL 03-3508-2091

■7月15日(水)～7月17日(金)
CASEジャパン'92
CASEツールによるソフトウェア開発・生産技術の展示会, 基調講演, 専門セミナーも同時開催される.
会場：サンシャインシティコンベンションセンターTOKYO (東京・池袋)
入場料：3,000円
問い合わせ先：リードエグジビションジャパン(株)
TEL 03-3349-8501

■7月21日(火)～7月23日(木)
ネットワーク'92
LAN, WANなどのネットワーク関連機器とアプリケーションの展示会, コンファレンスも同時開催される.
会場：日本コンベンションセンター (幕張メッセ)
入場料：1,000円
問い合わせ先：(社)日本パーソナルコンピュータソフトウェア協会　TEL 03-3221-7481

●セミナー

Microsoft University

■7月6日(月)～7月10日(金)
Microsoft Windows Programming Environment
Windowsアプリケーション開発のための入門コース.
受講料：180,000円 (5日間)

■7月6日(月)～7月10日(金)
MS OS/2 Applications Programming
MS OS/2アプリケーション開発のための上級コース.
受講料：180,000円 (5日間)

■7月13日(月)～7月17日(金)
MS OS/2 Presentation Manager Programming Environment
OS/2 PMアプリケーション開発のための入門コース.
受講料：180,000円 (5日間)

■7月13日(月)～7月17日(金)
Microsoft Windows Applications Programming
Windowsアプリケーション開発のための上級コース.
受講料：180,000円 (5日間)

■7月20日(月)～7月21日(火)
C言語入門
Quick Cを使ったC言語の入門コース.
受講料：60,000円 (2日間)

■7月22日(水)～7月24日(金)
C言語応用
MS-C 6.0を使ってC言語の高度な技術を学ぶ実習中心のコース.
受講料：90,000円 (3日間)

Microsoft Universityは, OS/2, Windowsアプリケーション開発者のために定期的に開催されているセミナーです.
問い合わせ先：富士ソフトウエア(株)MU教育センター
TEL 03-5461-8296

NEWS & NEW PRODUCTS

ビジネスショウ'92開催．WindowsMME，ペンコンピュータなどが出展

ビジネスショウ'92が，今年も晴海の東京国際貿易センターで開催された．今年は各社とも，「ダウンサイジング」と「ネットワーク」をキーワードとしたテーマ展開を行っており，例年に比べハードウェアの目新しさは感じられなかった．

昨年のビジネスショウで話題となったMS-Windows 3.0は，より高速なCPUや高解像度表示で動作するようになり，グラフィック，サウンド，アニメーションなどを追加したMME（マルチメディア拡張）によるデモがめだっていた．またDOS/Vの登場によってIBM互換機メーカーのブースの前で足を止める見学者も多く見られた．

ペン入力コンピュータは，先日発表した日本NCRだけでなく，三洋電機や沖電気工業が参考出品をしていた．どの機種もサイズはほぼ同じで，OSとして「Microsoft Windows for Pen」に標準で対応し，ほかにもゴー社のPen Pointに対応を予定しているとのこと．年内には各社から製品発表されそうな気配である．

東芝，松下電器産業などは，より高速化したラップトップ型のWSを展示していた．またネットワークの中核としてUNIXワークステーションを位置づけているメーカーも多かった．シャープは，DOS/V対応のパーソナルワープロ「書院パソコンPC-WD1A/B」，手書き入力を採用した電子システム手帳「ハイパー電子マネージメント手帳」を展示．とくに書院パソコンが入場者の注目を集めた．プリンター体型のデザインで，ワープロ機能を備えた上にOADG仕様のDOS/Vマシンとしても使える．

エプソンは先日発表した「PC-486GR」の専用拡張ビデオボードでのデモを行っていた．このビデオカードはWindows専用で，解像度は640×480と1024×768である．展示されていたカードには，IBM PC互換機用のビデオカードで使われているS3のチップが搭載されていた．

SONYはCD-Iプレイヤー「INTELLIGENT DISKMAN」，手書き入力電子システム手帳「PalmTop」，EBXA準拠の電子ブックプレイヤー「データディスクマン」，ポータブルファクシミリなど電子文具を数多く展示していた．また同社のAXパソコン上で「Windows NT」のデモも行っていた．



日本電気，低価格32ビットマシン，Windows MMEを発売

日本電気(株)，日本電気ホームエレクトロニクス(株)（NECパソコンインフォメーションセンター：TEL 03-3452-8000，N5200テクニカルセンター：TEL 03-3456-5200）は，PC-9800シリーズの新製品として，CPUを32ビット化，最大15%の低価格化を実現したデスクトップパソコン「PC-9801FX」と，処理速度を1.25倍，低価格化をはかった「同FS」の2機種を発売，TFTカラー液晶ディスプレイ搭載型ラップトップパソコン「N5200モデル03LC」に，CPUの高速化，低価格化などの強化を行った．「PC-9801FX」は16ビットパソコン「PC-9801DX」，「同FS」は32ビットパソコン「同DS」のそれぞれ後継機として位置づけられる製品で，現行機にくらべて最大15%の低価格化を実現している．

また同時に，「日本語MS-WINDOWS（Ver.3.0/3.0A）」に組み込むことにより，マルチメディアプラットフォームを提供する「日本語MS-WINDOWSマルチメディアエクステンションVer.1.0」を発売した．国内ではじめて，ビデオボードを利用して外部のビデオ機器から入力したビデオ映像をウィンドウ画面内へ表示させる「ビデオオーバーレイ機能」に対応，サウンド機能，アニメーション再生機能，複数のマルチメディアデータを統合するRIFFファイルをサポートしている．価格は12,800円，6月25日出荷開始の予定．

■PC-9801FX2，FX5，FX，FX/U2，FX/U5

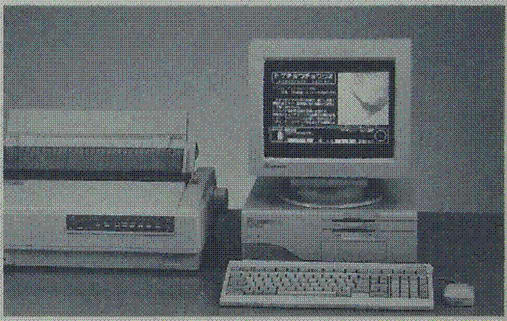
ワンラインNEWS
インターコムは給与計算ソフトの出力データを容易に銀行センターへ送信する給与振込ソフト「E-Bank」を開発．販売を開始した．

NEWS & NEW PRODUCTS

プロセッサ：80386SX（12MHz）／メモリ：1.6MB（最大14.6MBまで拡張可能）／外部記憶装置：5インチFDD×2（FX2），5インチFDD×2・40MB HDD×1（FX5），3.5インチFDD×2（FX/U2），3.5インチFDD×2・40MB HDD×1（FX/U5）／価格：278,000円（FX2，FX/U2），398,000円（FX5，FX/U5）

■PC-9801FS2，FS5，FS7，FS/U2，FS/U5，FS/U7

プロセッサ：80386SX（20MHz）／メモリ：1.6MB（最大14.6MBまで拡張可能）／外部記憶装置：5インチFDD×2（FS2），5インチFDD×2・40MB HDD×1（FS5），5インチFDD×2・100MB HDD×1（FS7），3.5インチFDD×2（FS/U2），3.5インチFDD×2・40MB HDD×1（FS/U5），3.5インチFDD×2・100MB HDD×1（FS/U7）／価格：348,000円（FS2，FS/U2），468,000円（FS5，FS/U5），538,000円（FS7，FS/U7）



エプソン，表示速度を向上させた初の486デスクトップパソコンを発売

セイコーエプソン（株），エプソン販売（株）（エプソンPCインフォメーションセンター：TEL 03-3377-3531）は，同社ではじめて80486SXを搭載し，表示速度を大幅に向上させたデスクトップパソコン「エプソンPC-486GR」，「同GF」を発売した．Windows環境をより快適にし，将来，パソコンを買い換えることなくアップグレードが行えることを基本コンセプトに開発されている．「PC-486GR」は，「PC-386GS」の後継機で，ハイレゾボードを標準装備している．「同GF」は「同386GE」の上位機にあたり，80486SX（16MHz）を搭載しながら34万円台の価格を実現した普及機．両機種とも，快適なWindows環境を提供するために，新開発の高速ビデオASICを採用，それぞれPC-386GS，GEとくらべて，テキスト表示で約3倍，グラフィック表示で約2倍の高速表示を実現した．

また同時に，エプソンPCシリーズ用OSとして，「日本語MS OS/2 Ver.1.21」と「日本語MS LANマネージャ Ver.2.0」を発売した．

■PC-486GR-2，2E，3，5，5E

プロセッサ：80486SX（25MHz）／メモリ：1.6MB（ハイレゾ時1.5MB，最大14.6MBまで拡張可能）／外部記憶装置：3.5インチFDD×2（2），3.5インチFDD×2・100MB HDD×1（2E），3.5インチFDD×2・5インチFDD×1（3），5インチFDD×2（5），5インチFDD×2・100MB HDD×1（5E）／価格：458,000円（2，5），483,000円（3），608,000円（2E，5E）

■PC-486GF-2，2E，3，5，5E

プロセッサ：80486SX（16MHz）／メモリ：1.6MB（ハイレゾ時1.5MB，最大14.6MBまで拡張可能）／外部記憶装置：3.5インチFDD×2（2），3.5インチFDD×2・100MB HDD×1（2E），3.5インチFDD×2・5インチFDD×1（3），5インチFDD×2（5），5インチFDD×2・100MB HDD×1（5E）／価格：348,000円（2，5），373,000円（3），498,000円（2E，5E）

日本IBM，PS/55 OS/2導入済みモデルを発売

日本アイ・ビー・エム（株）（TEL 03-3586-1111）は，「OS/2 J2.0」をあらかじめ導入したPS/55ファミリーの新モデルと関連製品を発売した．OS/2導入済みモデルとして発売されたのは，新モデルのモノクロ・ノートPC「PS/55noteモデルN51SLC」（8551-SZ8システム装置）と既発表のカラー・ノートPC「PS/55noteカラー・モデルN27sx」（5527-UZ8システム装置，価格：1,078,000円），高

NEWS & NEW PRODUCTS

機能デスクトップPC「PS/55モデル5551-W」(5551-WYA/WYB/WJAシステム装置，価格：990,000円～1,060,000円）の3機種で，OSとハードウェアを別々に購入するよりも，最大11%安く，またOSを導入する手間も省くことができる．

また同時に，低価格デスクトップ型DOS/V PC「PS/55シリーズ」の最上位モデル「PS/55Zモデル5530U SLC」を発表した．現行モデル（80386SX/20MHz）にくらべ処理能力を80%向上させた「386SLCマイクロ・プロセッサ」を搭載している．価格は540,000円～798,000円．

■PS/55noteモデルN51 SLC(8851-S08/SV8/SZ8システム装置)

プロセッサ：80386SLC(16MHz)／メモリ：4MB(S08/SV8,最大12MBまで拡張可能)，12MB(SZ8)／外部記憶装置：3.5インチFDD×1･80MB HDD×1／ディスプレイ：CFLバックライト付16階調モノクロ液晶ディスプレイ／価格：498,000円(S08,OS未導モデル)，514,000円(SV8,DOS J5.0/V導入モデル)，664,000円(SZ8,OS/2 J2.0導入モデル)

東芝，初の480SX搭載カラーブック型パソコンを発売

（株）東芝（パソコン・ワークステーション事業企画室：TEL 03-3457-2930）は，ブック型パソコンとしては国内ではじめてCPUに80486SXを搭載し，従来機（DynaBook V 386 J-3100SX）にくらべ，約2倍の処理能力を実現したA4ジャストサイズのカラーブック型パソコン「DynaBook V 486 J-3100XS 081VTモデル」を発売した．VGA対応の8.4インチTFT方式カラーディスプレイを採用，最大18万色中16色または256色を同時に表示できる．標準で4MBのメモリ，着脱可能な80MBハードディスクを内蔵している．

■おもな仕様

プロセッサ：80486SX（16MHz）／メモリ：4MB（最大12MBまで拡張可能）／外部記憶装置：3.5インチFDD×1・80MB HDD×1／ディスプレイ：FLサイドライト付きTFT方式カラー液晶ディスプレイ／価格：898,000円

クラスター，AT互換機対応のバックアップテープストリーマーを発売

（株）クラスター（TEL 03-3257-3261）は，IBM PC/AT互換機，DOS/Vマシン専用HDDバックアップテープストリーマー「プロップセーバー」を発売した．空いている5インチベイに挿入し，フロッピーケーブルと電源ケーブルをつなぐだけのビルトインタイプで，1分間に約3MBを高速に圧縮しながら120MBテープにバックアップする．DOS用，Windows用の英語モード用プログラムとQic80フォーマット済み120MBテープカートリッジが付属している．価格は64,800円．

マイクロソフト，98NL対応のメモリカード版Worksを発売

マイクロソフト（株）（インフォメーションセンター：TEL 03-3363-1201)は，ワープロ，表計算，データベース，通信の4つの機能を統合した「Microsoft Works Ver.2.5」を1枚のメモリカードにおさめ，PC-9801NL（98NOTE LIGHT）対応版として発売した．すべてのツールの操作性は統一され，各ツール間でのデータのやりとりも自在に行える．学習ソフトも標準装備しており，初心者でもすぐに使いこなすことができる．価格は40,000円．

ボーランド，98・J-3100対応dBASEIV新バージョンほかを発売

ボーランド（株）（営業部：TEL 03-5350-9370）は，パソコン用リレーショナルデータベースソフ

ワンラインNEWS
シーイーシーは従来から販売していたカード型DBソフト「Q&A」に画像処理機能を追加した新バージョンを発売した．

NEWS & NEW PRODUCTS

ト「dBASEIV 1.1J」を発売した。これはPC-9800シリーズ対応の新バージョンで，メモリ管理方法に従来のVCPIに加え，DPMIとXMSを追加，さらに対応機種にPC-9800シリーズのハイレゾモードとEPSON PC-286/386シリーズを追加した。また新たに，80286CPU以上の全機種を対象とし，カラーディスプレイモードにも対応したJ-3100シリーズ対応版，機種依存をしないユーザー管理方法を採用し，5インチと3.5インチを同梱したマルチベンダー版のLAN拡張用パッケージ「dBASE IV LANPack」を発売した。「dBASEIV 1.1J」は，新たに採用されたユーザーインターフェース「コントロールセンター」から，追加・修正・削除といった通常のデータ処理をはじめとして，画面設計やラベル／レポートのデザイン，アプリケーションの開発まで，ほとんどの操作を行うことができる。１つの画面上でファイルの連結，データの抽出などができる「クエリー」，メニュー操作だけでアプリケーションプログラムの作成ができる自動プログラム作成機能「アプリケーションジェネレータ」などを備えている。価格は「dBASEIV1.1J」は198,000円，dBASEIVLANPackは，298,000円（5ユーザ版），477,000円（10ユーザ版），1,550,000円（40ユーザ版）。

エム・ピー・エクノロジー，Windows用アプリケーションを発売

（株）エム・ピー・テクノロジー（営業部：TEL 03-3358-9800）は，Windows 3.0に対応したカード型データベース「Radish Ver.3.0」を発売した。マウスで簡単にフォームを作成することができ，作成後のデータエリアのサイズ，位置，属性などの変更も容易に行うことができる。また，Windows 3.0のペイントブラシや同社の作図ソフト「Tomato Ver.3.0」などで作成したグラフィックデータをクリップボード経由で各カードに貼り込むことができる。価格は28,000円。

また同社では，簡単にWindowsのアイコンやカーソルを好みのデザインに変更することができるユーティリティソフト「Icon Do-It!」を７月に発売する。200種類のデザインアイコン，15種類のアニメーションアイコン，５種類のアニメーションカーソル，４種類のスクリーンセーバーが用意されている。価格は9,800円（予価）。

吉正電子，OS/2対応のユーザーサポートシステムを発売

吉正電子（株）（PC事業部：TEL 03-5371-3041）は，パソコン通信を利用したOS/2対応のユーザー・サポート・システム「PC-NET/2 for OS/2」を発売した。製品購入者に対するアフターサービスを目的としたシステムで，製品案内や問い合わせ，バージョンアップの情報交換などに利用することができる。メニューはテキストファイルで提供されているためカスタマイズが容易。一般的なBBSソフトウェアとしても利用することができる。全ソースコードを付属し，ユーザーアプリケーションへの組み込みや通信による情報交換など，ニーズにあわせたネットワーク構築が可能。価格は198,000円。

アルパソフト，文書編集機能をもつWin対応カード型DBソフトを発売

アルパソフト（株）（TEL 03-3863-5976）は，カード型データベース機能，文書編集機能をもつWindows Ver.3対応の「Alpasoft Estella Ver.1.0」を発売した。カード型データベースは，リスト型，T字型，カード型，併用型の入力タイプをもち，ビットマップイメージもデータとして扱うことができる。文書編集機能は，編集モードとレイアウトモードをもち，文書作成とレイアウトを行うことができる。またタスク切り替え，ディレクトリ表示やファイルの複写，削除などが行えるファイル管理機能も備えている。価格は58,000円。

Programmer's Program
アセンブラソースジェネレータ
SOURCER Ver.4.5

「SOURCER」は，PC/AT用ソースジェネレータとして米国では絶大な人気を誇っている．日本でも昨年から(株)ライフボートによって販売され，すでにソースジェネレータの定番として定着している．このSOURCERが新バージョンとなって登場した．

追加された新しい機能

新バージョンになったSOURCERは，Microsoft Macro Assembler Ver.4.0/5.0/5.1，Borland Turbo Assembler Ver.1.0/2.0，OPTASM Ver.1.0/1.5に加え，Microsoft Macro Assembler Ver.6.0，Borland Turbo Assembler Ver.2.5/3.0といった最新アセンブラに対応した．さらに，MS-DOS 5.0で新たにサポートされたファンクションリクエストはもちろん，DPMIやVCPI，そしてWindowsファンクション（int20H VMMファンクション）のコメント出力もサポートした．また，従来のリアルモードデバッガでは解析が不可能だったWindows VxD(仮想デバイスドライバ）に組み込まれた32ビットフラットなプログラムコードを自動検知し，最新の32ビットアセンブラコードに変換することも可能になった．

ほかにもハードウェアテーブル機能をサポートすることで，公開されていない未知の割り込み処理やI/Oポートアクセスに関するコメントをユーザー自身が定義することもできるようになり，さらに文字列を参照しているコードのコメント部にそのテキスト内容をインライン表示することができるようになった．

以上のように，多機能になったにもかかわらず，解析速度は従来のバージョンに比べ，約5倍から最高10倍までと高速化を実現している．

PC-9801版を用意

従来のSOURCERは，ソフトウェア割り込みやI/Oポートアドレスを検出したときのコメントはPC/ATのハードウェアに準拠したもので，PC-9801がターゲットマシンの場合はあとでデータブックなどで調べ直す必要があった．しかし，Version 4.5ではPC-9801版（以後98版）を用意することで，PC-9801のハードウェアに依存したソフトウェア割り込み（BIOSコール）やI/Oポートアクセスに関する手続きを検出，解析ソースのコメント部，あるいはクロスリファレンステーブルに発見された手続きをレポートすることを可能にした．また98版とすることで，画面もエスケープシーケンス制御からVRAMの直書きへと変更になり，見やすさと表示の高速化が図られている．

なお，SOURCERのパッケージは98版・FMR／J-3100／AX版・PS/55（DOS/V）版・PC/AT（英語DOS）版がパックになっており，インストール時に使用機種を選択する．98版とPC/AT版以外の機種では，エスケープシーケンス制御に専用の表示ドライバが必要になる．これがDOSにデフォルトで組み込まれていない機種では，あらかじめDOSのシステムディスクに標準で添付されている表示ドライバ（ANSI.SYS，ANSIJ.SYSなど）をCONFIG.SYSに追加しておく必要がある．また，国内で急速に普及しているDOS/Vにも対応していながら，同じPC/ATでの英語DOS版の画面と違って，エスケープシーケンス制御の単色画面であるのが，残念だ．

基本操作

SOURCERを使用するには，MS-DOSのコマンドラインから

A>SR ［解析ファイル名｜定義ファイル名］［オプション］

と入力する．定義ファイルが指定されてる場合，これを初期化情報として読み込み，メインメニューを表示する．コマンドラインからファイル名を省略した場合は，ここでファイル名入力プロンプトが表示されるので，解析ファイル名を入力する．画面に表示されているハイライト表示（98版の場合は反転）の英字記号は，各種コマンドに対

応している．

EMSメモリがあれば，SOURCERは，それを自動的に認識し使用する．これにより，大きなファイルのロードが可能となり，プログラムの解析中に生成されるシンボルのために十分な領域を確保することができるようになる．SOURCERは，最大6MBのEMSと100個のハンドルを使用する．ハンドル数を指定するオプションのあるEMMドライバの場合は，最適なパフォーマンスを発揮するためにも，ハンドルを最低でも40確保するよう勧めている．

プログラム中の日本語文字列

起動時-jオプションを指定すると，全角文字列を処理できるようになる．ただし，全角文字列とコード（命令）の判断は難しく，時間がかかる上に解析精度が低くなる．回避方法として，特定のアドレスに全角の文字列が入っていることが前もってわかっている場合には，定義ファイルに文字列指定を行うことを勧めている．

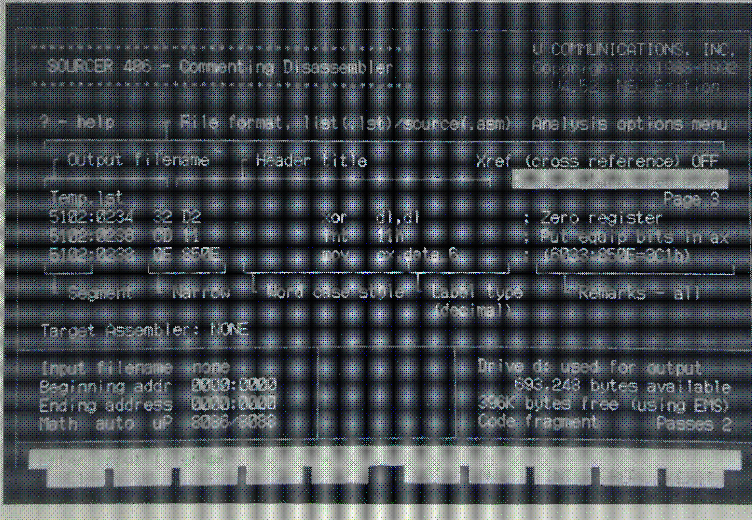
なお，このオプションはプログラムに全角文字列を含まないことがわかっていれば，指定する必要はなく，PC/AT（英語DOS）版では無効である．

出力されるアセンブラソース

解析されたアセンブラソースは，任意の処理系に依存した形式で出力でき，8088から80486までのすべてのCPUに対応している．もちろん，数値演算用コプロセッサ用のコードも解析する．そして使用されているファンクションリクエストの機能と各レジスタの意味，I/Oポートの機能，複雑なコード処理内容などについての，詳細なドキュメントに類するコメントが付加される（BIOSとI/Oポートについての解析は98版およびPC/AT版のみ，またコマンドによってコメントの出力を抑制することもできる）．

このほかに，「.SDF」という拡張子のついた，解析時に検出された各種アイテムの定義ファイルも出力される．これを変更し，DEF（定義ファイル）として SOURCERに与えることで任意の解析方法をSOURCERに指定することができる．なお，従来のバージョンから解析オプションが変更になっているので，従来の定義ファイルを使用する場合は，意図する解析オプションが指定されているか事前に確認したほうがよいだろう．

従来からSOURCERは，486までのあらゆるCPUに対応し，MS-DOSの実行形式ファイルだけでなくデバイスドライバやバイナリファイルそして任意のアドレスのROMの内容まで，あらゆるプログラムを処理対象としていた．生成されるアセンブラソースはターゲットアセンブラを指定することで，もちろん再アセンブル可能だった．従来から強力かつ正確だったSOUCERが，PC-9801のBIOSおよびI/Oポートのサポートをした今回のバージョンアップの意味は大きい．またユーザーインターフェースの改良など，着実な進化も見逃せない．



SOURCER Ver.4.5

販売：（株）ライフボート（TEL 03-3293-4711）／価格：38,000円／動作機種：日本電気PC-9800シリーズ，富士通FMRシリーズ，東芝J-3100シリーズ，各社AX，日本IBM PS/55シリーズ（DOS/V），IBM PC/ATおよびその互換機（英語DOS）／動作環境：MS-DOS Ver.2.xx以上／必要メモリ：640KB以上／供給メディア：5インチまたは3.5インチフロッピーディスク／日本語マニュアル

オンラインソフト採集帳 19

3Dドライビングゲーム Hover Run

「Hover Run」は，ワイヤーフレーム描画による3Dドライビングゲームだが，自動車でなく宙に浮くホバーカーを使うのが特色だ．ゲームには，最大18人のプレイヤーが参加できる．ただし，プレイヤーは同時にコースを走るのではなく，それぞれ12のコースを走り，ベストタイムによるドライバーズポイントを競うことになる．このゲームの動作機種は，日本電気のPC-9800シリーズまたはその互換機．CPUが286以上の機種でないと遅すぎてゲームにならない．また，BASIC ROMが装備されていない一部の互換機では動作しないので注意が必要である．

ゲームディスクの作り方

Hover Runは，N88-BASIC上で動作するゲームのため，MS-DOSから立ち上げることはできない．まず，専用のゲームディスクを作成する必要がある．ゲームディスクを作成するには2HDのFDD 1ドライブを含めてHDDやRAMディスクなど2ドライブが必要となる．ただしゲームディスク作成後は，ほかのゲームディスクからプレイヤーデータのコンバート以外は2HDのFDD 1ドライブで動作可能である．

ドライブ番号1のFDDに未使用のフロッピーディスクを挿入し，ハードディスクやRAMディスクなどカレントに

```
HOVER.DSK
HOVER.BAT
DISKLZ.EXE
```

の3つのファイルがあるのを確認した上で，MS-DOSのコマンドラインから

```
A> HOVER
```

と入力すれば，ゲームディスクが作成される．このバッチファイルはディスクファイル化ツール「DISKLZ.EXE」を呼び出し，ファイル化された「HOVER.DSK」をドライブ番号1のFDDに挿入されているフロッピーディスクに対して復元しているだけである．

DISKLZはファイルをディスクに復元する際，FDDのドライブ番号1に対してしか指定できない．つまり，内蔵FDDが5インチの機種で外付けの3.5インチFDDに対してゲームディスクを作成するにはあらかじめ，FDDのドライブ番号を入れ換える設定を行う必要がある．

起動装置が標準に設定してあれば，ゲームディスクをドライブ番号1のFDDに挿入し，リセットするだけでHover Runが起動する．なおゲームディスクはゲーム中に抜いたり，入れ替えたりすると動作しなくなる場合もあるので，注意が必要である．

操作方法

はじめてプレイする場合は，まずプレイヤーエントリーを行う．これはトップメニューから［↓］（または［↑］）を押して「User Entry」を反転させ［RETURN］を押す．プレイヤー名の入力画面となるので，キーボードを使って入力する．ただし全角文字は使用できない．

プレイヤーのエントリーが終了したら，いよいよゲームの開始である．トップメニューから「Start Game」を選択し，プレイヤー（Player Select）・コース（Course Select）・車（Car Select）を選択し，スタートである．コースは全部で12種類が用意されていて，好きなコースを選ぶことができる．また車は3種類から選択することができる．ただし運転するのは地面を走行する「自動車」ではなく，宙に浮いて走行するホバーカー（Hover Car）のため慣れが必要である．各メニューでの基本操作は［↓］／［↑］で反転（カーソル）の移動，［RETURN］で決定，［ESC］で前のメニューに戻ることができる．

ホバーカーの運転は［4］／［6］／［SHIFT］／［XFER］を使用する．［4］で左，［6］で右へホバーカーの転回，［SHIFT］でアクセル，［XFER］で逆噴射（ブレーキ）である．

タイムアタック

ルールはコースを2周し，そのタイムによって与えられるポイントを競う（写真2）．走行中はコースを外れると跳ね返され，機首がコース内に向いていないとアクセルを踏んでも前方に進むことができないので，転回または逆噴射してコースに戻る必要がある．なお走行中は［STOP］キーを押すことで，リタイアすることができる．

記録は自己ベストとコースベストというのがあり，自己ベストを更新すると記録タイムと順位が上がり，コースベストを更新すると当然1位となって，ベストの走りが登録される．

ビデオモード

ゲームが終了したあとはビデオモードになり，これで直前の自分のホバーカーの走行を見ることができる（写真3）．ビデオモードの操作はテンキーを使用し，[2] で再生・[8] で逆再生となる．ほかにカメラアングルの変更も可能で，[4]／[6]で視点の移動，[+]で拡大，[-] で縮小する．

トップメニューから「Option」メニューを選択すれば，並走車の有無(Best Car)，エンジン音の消去(Beep) などの設定ができる．「Option」メニューは，ほかにもエントリーしたプレイヤーの削除やデータのコンバートを行うことができる．ただしプレイヤーデータのコンバートはあらかじめ同じ名前のプレイヤーが登録されている必要がある．

Hover Runはワイヤーフレームを使った3Dで，しかもモノクロ画面のためゲームセンターの3Dドライビングゲームのような派手さはない．しかしタイムアタックだけでなく，ビデオモードなど，変わった視点でゲーム後も十分楽しませてくれる．これで，MS-DOS上から起動できれば，遊ぶ回数が増えるのは間違いないだろう．

■Hover Run

名称：Hover Run／分類：ゲーム／登録名：HOVER_01.LZH／作者：Y. Nomura（野村佳秀）／動作環境：日本電気PC-9801シリーズ（VX以降）およびその互換機／復元方法：LHA E HOVER_01

ファイル構成

Listing of archive : HOVER_01.LZH

Name	Original	Packed	Ratio	Date	Time	Attr	Type	CRC
HOVER.DOC	5460	2616	47.9%	92-01-28	21:59:20	a--w	-lh5-	2C02
HOVER.BAT	427	271	63.5%	91-12-14	17:24:14	a--w	-lh5-	724E
HOVER.DSK	382714	245274	64.1%	91-12-25	18:12:26	a--w	-lh5-	C181
DISKLZ.EXE	7939	7673	96.6%	91-12-21	19:17:46	a--w	-lh5-	79C7
4 files	396540	255834	64.5%	92-02-20	21:37:36			

写真1　オープニング

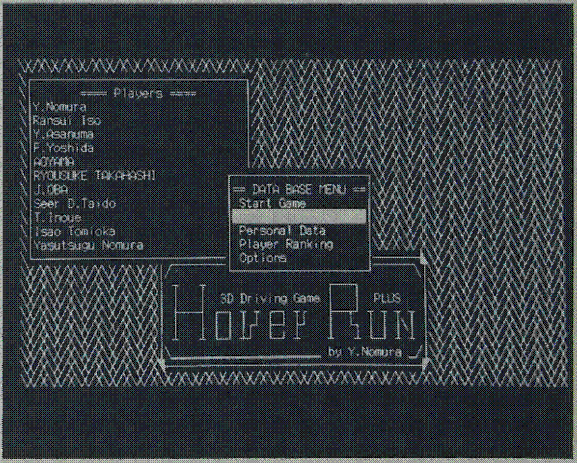


写真2　タイムアタック

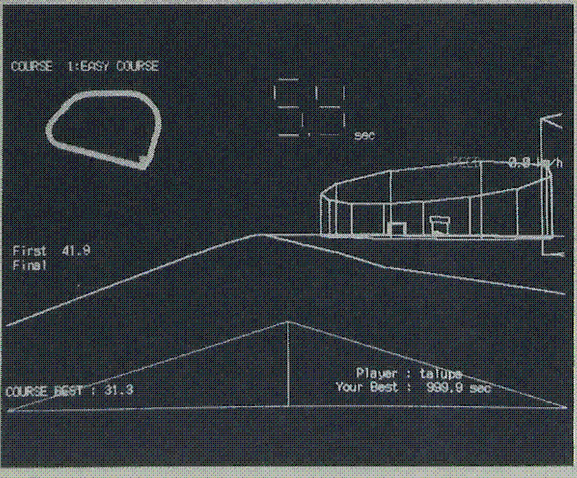
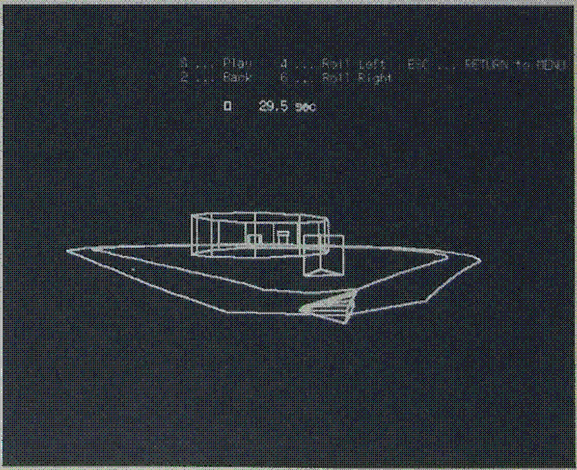


写真3　ビデオモード



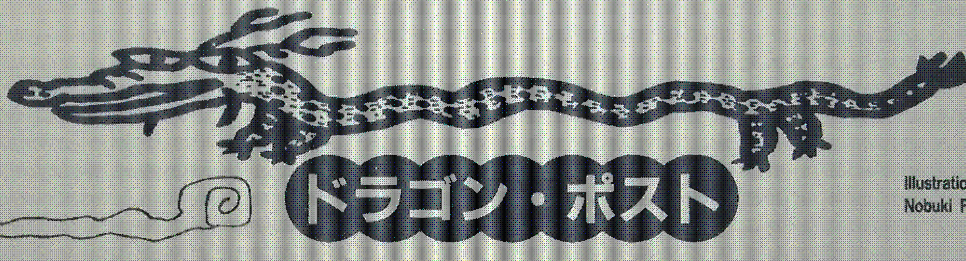


Illustration
Nobuki Fukuhara

この欄は，読者からのお便りを中心に構成します．本誌に対するご意見・ご感想などをお待ちしております．鬼より恐い読者からの声ということで，「ドラゴン・ポスト」です．本誌についているアンケートはがきにご意見の欄を設けてありますから，そこに書いてください．ペンネームでも歓迎です．ただし，ハガキには本名と住所をしっかり書いてくださいね．掲載誌をお届けします．

★計算機工学研はニューラルネットにあけくれるか？

友人：「ついに研究室に配属されたって？」
私　：「そうだよ．毎日いそがしくて……」
友人：「情報の計算機工学ってぐらいだから，ワークステーションでニューラルネットワークのシミュレーションなんかするんだろ？」
私　：「そんなのやってないよ」
友人：「でも，1日中プログラミングにあけくれてるんだろ？」
私　：「違うよ．OPアンプの半田付けだよ．アナログ回路って，なかなか理屈どおりに動いてくれなくって」
友人：「……」
彼が無口になったのはいうまでもない．(Ship)

――そんなもんですよね．外から見るのと，中にいるのとでは大違い．

★ソフトの充実のためには開発者を大切に

MS-C Ver.7.0がアメリカではリリースされたそうです．そしてこんどはC++対応とか，さらにWin3.1SDK付属とも，かなりBorlandの戦略を意識しているようですネ．きっと日本でも，日本に合わせた形でリリースされるでしょうが，はたしてユーザ（プログラム作成者）の立場を考えた構成と価格にしてくれるでしょうか？新しいOSや環境の普及はなんといってもソフトの充実．そしてそのためには多くの人がソフトを作らなければならないと思います．その点，日本では使う側のことばかりに目がいっていて，ソフトを作るほうはなおざりにされていると思いませんか？（佐野浩之）

――「C/C++ Development System for Windows 7.0」というのが製品名です．WindowsSDKも付属しています．驚いてしまうのは，パッケージの重さ．十数キロはあるという段ボールの箱にC/C++とWindowsSDKが納まっています．箱が大きいだけでなく，マニュアルがびっしり詰まっていて見かけより重く感じます．そして，導入に必要なHDの空きは50MB以上．先日バージョンアップされたBorland C++のパッケージといい，言語製品の巨大さに圧倒されてしまいます．ところで，日本でリリースされるC7.0にSDKは付くのか？　日本語Windows3.1の出荷がいつになるかのかねあいがあると思うのですが．

★ああ！時の流れのすさまじさ……

念願のIBM PC/AT互換機を30万円弱で買った．思えば数年前買ったMZ-2000．GRAMを3ページつけ（なければグラフィックが表示できなかった），カラーディスプレイを購入，金額はMZ-2000の方が高い!!　それにもかかわらず性能は……．あのころのパソコンが高価だったのか，現在のパソコンが安いのか，時代の流れはすさまじいと思う今日この頃です．（村山隆行）

――これはもうしょうがないですね．パソコンを買ってしまうと必ずこう感じるときがあるはずです．ひどいときは，買った翌日にコストパフォーマンスの高い新製品が発表されることもあります．何年もたつと性能など比べようもないということになってしまいます．

★香港へ個人輸入旅行

PCコンパチ機ほしさに構想2カ月，実行1週間の個人輸入旅行をした．そこは香港，コンピュータパラダイス．すごいの一言．詳しいことを書くスペースはないので書けないが，98を買うくらいなら，香港に買いに行った方が絶対いい．おすすめだよ．P.S.香港のマックのハンバーガーはHK$4，約70円．（大上茂雄）

――香港でPCを買う話は，本号でもレポートしています．行った人はみんな満足して帰ってくるようです．ところで，ハンバーガーのマクドナルドは東京では「マック」といいますが，関西に行くと「マクド」と呼ぶというのは本当ですか？

★あのPC-8201が現役バリバリで活躍!?

5月号のドラゴンポストの時間蝿さんへ．
私は19歳ですが，PC-8201を知っています．なぜって，なんと！私がバイトしていた先では，1992年4月現在，PC-8201＋バーコードリーダーが3台（もっとあるかも知れない）が，現役バリバリで働いているんです．すごいでしょう．（斉藤吉彦）

――うーん，すごい．あのPC-8201は，コンパクトに作られていて持ち運びしやすいから，オフライン端末にして

データを吸い上げるなんて使い方ができるのでしょうね．なんにしても，活躍しているのならいいことです．

★Cの教育をやれと業務命令が

C＋＋を仕事で使い始めて，4カ月になります（Cもさほど理解していないのに……）．そんな私に突然，課員に対してC言語の教育をするようにと，業務命令が下りました．えーと，関数を理解させて，ポインタ，配列，構造体，リスト構造……（頭が痛い）．さぁーて，今月の特集を参考にして，C言語の勉強し直すかなぁー．（朝野一彦）

――「特集：Cの学び方・教え方」．グッドタイミングでしたね．少しはお役に立てたでしょうか．

★おちついてパソコンを使う日はいつ？

自宅にパソコンが入り，1年．TURBO C＋＋を買って半年がすぎ，会社から帰ると，パソコンにとはいかず，子供（5カ月）をふろに入れ，ご飯を食べ，9時頃パソコンに……．あっ，眠気が……．気がつくと12時ごろ，パソコンの電源を入れるチャンスをのがす．いつになったらおちつくやら……．（KOJI）

――あー，これはたいへんだ．もうすぐお子さんが歩き始めます．子供はパソコンが大好きです．キーボードでも，電源スイッチでも，動くものはさわろうとします．しばらくは，お子さんが起きているうちはパソコンにさわれません．さらに，もっと動き回るようになると，お父さんが出勤している間にドライブに何かを突っ込まれたりします．十分注意しましょう．(^^;)

★FEPは大きくなっていく割に……

ワープロソフトでこれといったものがないと感じる．特にFEP部分がどんどんメモリ消費型になっていき，大きくなった割にメリットがないように思えるのですが，最近のソフトはハードに甘えているのでは？最新のハードを持っている人間ばかりとは限らないし，バブルの崩壊で企業も新機種を買いひかえるときなのに……と思う．（奥谷文隆）

――FEP開発者は，Windows対応を迫られています．Windowsが動作する環境を導入すると，高速なCPUと大量のメモリが必要になります．640KBの壁であくせくしていた頃のことは忘れて，新しい環境でFEPを開発します．おまけにWindowsプロセスで動作する日本語変換プログラムはメモリの制約から解放されるため，サイズを抑えようという気持ちはどこかへいってしまいます．その結果として，FEPは大きくなってしまう．と，考えてみたのですが……ホントかな？

★プログラムがぐちゃぐちゃになるのはどうして？

最近HDの中が混雑してきまして，バックアップはしているものの，整理の仕方が悪いのか，時折データを謝って消すことがあり，バックアップディスクを探すのに半日以上（笑）もかかり悩んでいます．性格上のこともあって，あれやこれやプログラムを組むものの，しり切れとんぼのようになっているプログラムをAとBとMERGEするとき，誤ってぐちゃぐちゃにするのは，毎晩夜中の2，3時までプログラムを作っているせいでしょうか？起きて会社にいくのが7時半なんて少しハード（？）でしょうか．（唐崎豪志）

――「誤ってデータを消す」「誤ってぐちゃぐちゃにする」，さらに「性格上のこと」があって，おまけに睡眠時間が4，5時間．かなりハードな内容のおハガキでした．(^^;)　「おねえちゃん，生活変えたら」というゴクミのCFが思い出されます．:-)

★「永久保存版」LDの寿命は？

僕も「ファンタジア」LD（限定版）を買っているのですよ．永久保存ってことだったのですが，LD（CDも同じ？）の寿命が確か10～15年程だったと思います．CD（LD）のユーザーのもつライセンスってのはどの程度なんでしょうね．記録面の酸化膜が劣化してまともに聞けなくなった時，（たとえ有料でも）交換を保証してくれるのでしょうか．どう考えても交換してくれることなどないように思います．ということは，多少のノイズがあっても，レコードやビデオを買った方が良かったのでしょうかね．といっても，ビデオはあってもレコードなんかもうないしね……．ダビングして保存するしかないのかなぁ……．（KID）

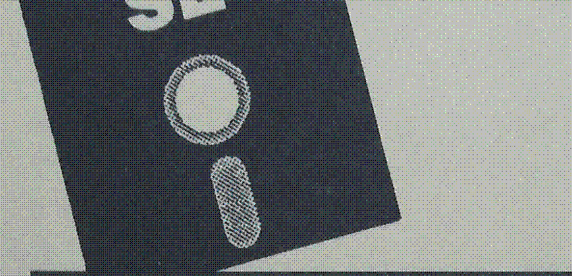
――LDは，ビデオカセットに比べて画質がいい，再生を重ねても画質が劣化しないというのが売りだったんですが，最近は10年寿命説が話題になっています．実際に酸化膜が劣化して再生できなくなったという例はあるのでしょうか．

★100MBハードディスクが30万円？

必需品（HD）を買うために，今も一れつに働いている．この分だと，2カ月後には100MBのSCSIが手に入りそうです．ところで，友達に「今100MBのHDっていくらだ？」と質問したら，「30万円位かな？」だって．私の調べた金額の3倍ぐらいだと思っているようなので，「食い下がれば（値切れば）1ケタ万円で買えるよー」と教えると，「オレも買おうかな？」だってさ．（豊田眞一郎）

――バイトをしてハードディスクを買おうという学生さんですね．7月号が出るころにはもうSCSIハードディスクを手にしているかな？

（敬称略）



付録ディスクのご案内

このフロッピーディスクに含まれる実行ファイル，ソースリストおよびテキストファイルは「月刊Programmer's Page」1992年7月号に掲載されたものです．

付録のフロッピーディスクは，5"2HD（8セクタ／トラックMS-DOS）でフォーマットされています．3.5"や2DDなどへのコンバートサービスは行っておりません．なお，オンラインサービスSE-NET（03-5276-1297）でも，同じ内容のファイルを用意しています．

フロッピーディスクの内容は次の通りです．本誌およびディスクに含まれるドキュメントファイルをよく読んでからご利用ください．

■ディスクのおもな内容

●付録ディスクについての説明

ディレクトリ名：ルート

収録ファイル名：

README.BAT（ディスク内容の説明を読むためのバッチファイル，PC-9801専用）

README.DOC（ディスク内容の説明，MS-DOSテキストファイル）

ONLINE.BAT（オンラインソフトの説明を読むためのバッチファイル，PC-9801専用）

ONLINE.DOC（収録されているオンラインソフトの簡単な説明）

V.EXE（ミニページャー，PC-9801専用）

BOOK.TXT（翔泳社書籍目録）

●プログラム「遊」工房（p.62）

「七並べ」ゲーム

ディレクトリ名：PLAY

収録ファイル名：PLAY.LZH

●実践C塾　アルゴリズム篇（p.76）

応用・その1

ディレクトリ名：JUKU

収録ファイル名：JUKU.LZH

●98活用アセンブラ技法　キーボード編（p.86）

キーボードロック／リセットプログラムの作成

ディレクトリ名：98ASM

収録ファイル名：98ASM.LZH

●綴じ込み付録

ディレクトリ名：DATA

収録ファイル名：ESC_CHR.TXT（MS-DOSテキストファイル）

●オンラインソフト採集帳（p.136）

今月号で紹介した3Dドライビングゲーム「Hover Run」と，1つまたは複数のファイルの圧縮および「.LZH」という拡張子をもった圧縮ファイルの復元をする高圧縮復元書庫管理ソフト「LHA」が収録されています．LHAの復元方法についてはONLINE.DOCを参照してください．

ディレクトリ名：ONLINE

収録ファイル名：

HOVER_01.LZH（Hover Run，PC-9801用）

LHA213.EXE（LHA v.2.13，MS-DOS汎用）

●クイズ・ランダムスクリーン（p.142）

クイズ解読ツール「PUZLE」

本誌掲載のクイズ・ランダムスクリーンの画像をディスプレイ上に再現し，マウスを使って並べ換えることにより，解答を得ることができるユーティリティです．PC-9801シリーズ専用です．詳しい使い方はPUZLE.DOCを参照してください．

ディレクトリ名：OMAKE

収録ファイル名：

PUZLE.EXE（解読ツール，PC-9801専用）

PUZLE.DOC（PUZLEの説明）

CONFIG.SYS（PUZLE用CONFIG.SYS）

AUTOEXEC.BAT（PUZLE用AUTOEXEC.BAT）

RNDSCREN.LZH（画像データ）

TORA.LZH（5月号の解答データ）

TORA.BMP（Windows 3.0の壁紙）

●そのほか

福田洋氏のご好意によりLOGOインタプリタ「Ashi LOGO」を収録しています．対応機種は日本電気のPC-9801シリーズですが，C言語のソースファイルも収録されており移植は容易になっています．

ディレクトリ名：LOGO

収録ファイル名：ASHILOGO.LZH

※使用上の注意

本誌に掲載されたプログラムおよびデータを利用するには，MS-DOSの基礎知識が必要です．このディスクには，MS-DOSのシステムは入っていません．別途，MS-DOSが必要となります．また，ソースファイルのコンパイルなど，別途コンパイラやユーティリティが必要な場合があります．

■圧縮ファイルについて

ディスクスペースを有効に使うために，多くのファイルは圧縮した上で収録しています．「.LZH」という拡張子のついたファイルは，高圧縮書庫管理ユーティリティ「LHA」によって圧縮されたファイルです．これらのファイルを利用するには，まずLHAで復元する必要があります．LHAの利用方法については，ONLINE.DOCをご覧ください．

バーチャルスペースガイド

P子　某ソフトハウスで企画営業を担当している．26歳．
C平　某メーカーのシステムソフトウェア開発部門に所属．25歳．

P子　何をニヤついて歩いているのよ．また，エッチなビデオのことでも考えているんでしょう．
C平　へへへ．ビデオのことには違いないけど，昨日見たんだ．アカデミー賞の……何だっけ，ジョディ・フォスターが出てるやつ．
P子　何ぼけてるのよ．「羊たちの沈黙」でしょ．で，ジョディ・フォスターにいかれているっていうわけ．
C平　別に昨日いかれたわけじゃなくて，「ホテル・ニューハンプシャー」とか，みんなよかったな．
P子　うーん．漁師町のおばさんみたいな迫力があるのよね．私のまわりにもジョディ・フォスターが好きだっていう男の子が多いよ．あの迫力がマザコン男の気持ちに訴えるのかしら．
C平　ひどいいい方だな．確かに，ハードな役をこなす迫力はあるけど……．「ホテル・ニューハンプシャー」で，ロブ・ロウがやっている弟と兄弟喧嘩するシーンとか，いかにも気が強そうだというとこがいいよね．
P子　ほら見なさい．強い女に守って欲しいっていう気持ちがあるんじゃないの？で，肝心の映画のほうはどうだったの，私はけっこう面白かったけど．
C平　すごーくではないけど，面白かった．でも，人間の不気味さみたいなものを出すんならもっと突っ込んで欲しい気がするし，サスペンスのほうにいくんならもっと方法はあるように思うけど……．
P子　私は，あれ以上不気味であって欲しくないわ．ちゃんと見てたのかしら．

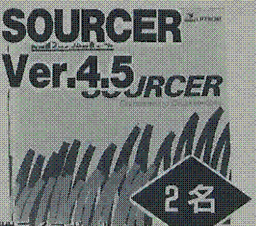
■羊たちの沈黙

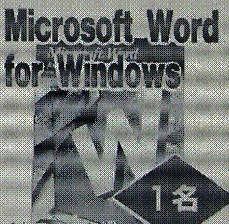
(1991年・米)
監督：ジョナサン・デミ
出演：ジョディ・フォスター
4,700円(税別)　パイオニアLDC



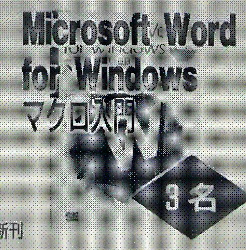
ジョディ・フォスター扮するFBIの実習生は，連続殺人事件の捜査のために服役中の殺人犯である精神家医に接見することになる．精神家医は事件の鍵を握っているらしい．その情報を引き出すために，彼女と精神家医の心理ゲームが展開する．その中で，議員の娘が誘拐され事態は切迫する．

読者プレゼント

1　SOURCER Ver.4.5

2名
提供：㈱ライフボート

2　Microsoft Word for Windows

1名
提供：マイクロソフト㈱

応募は本誌に添付のアンケートはがきをご利用ください．(1992年7月17日消印有効)．なお，応募者多数の場合は，抽選になります．当選者発表は発送をもって代えさせていただきます．

3　Microsoft Word for Windows マクロ入門

3名
翔泳社新刊

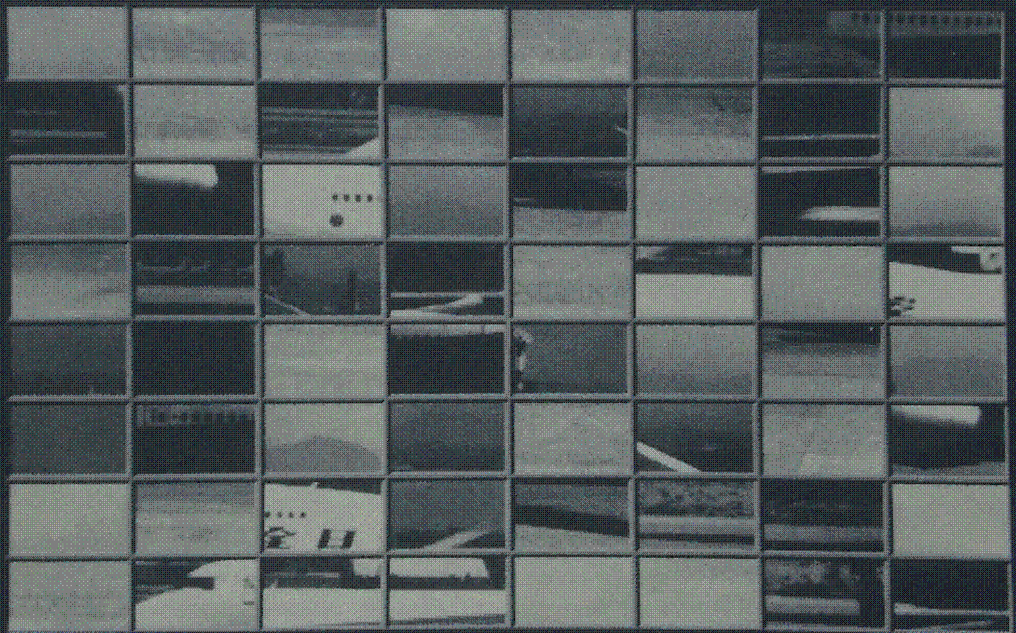
4　フリーソフトウェア GAME派宣言

3名
翔泳社・PC-PAGEシリーズ新刊

クイズ・ランダムスクリーン

ある１枚の画像を64分割し，ランダムに並べ替えてあります．
さて，もとの画像は何を写したものでしょうか．

ヒント：鳥だ！　○○○だ！



正解者の中から30名様に本誌表紙イラスト（日暮修一画）入り月刊プログラマーズ・ページ特製オリジナルマグカップをさしあげます．

応募方法：本誌に添付のアンケートはがきをご利用ください．

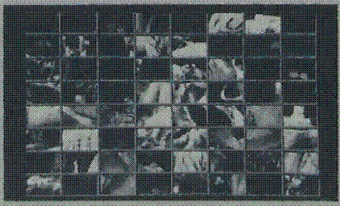
締め切り：1992年７月17日（消印有効）

なお，当選者の発表は発送をもって代えさせていただきます．

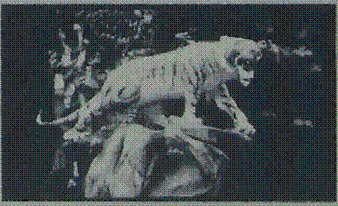
＊画像データは，付録ディスクやBBSでもサービスしています．

６月号の問題と正解

【問題】



【正解：とら】



SE-NETからのお知らせ

SE-NETでは開設以来，オンラインサービスを行ってきましたが，回線混雑のため非常に接続しにくい等の問題が出てまいりましたので，GUESTによるダウンロードは中止させていただきます．今後は読者のみなさまと編集部とのコミュニケーションの場としてご利用ください．

●BBSへの接続

SE-NET　03-5467-0376

通信速度	300/1200/2400bps
接続プロトコル	CCITT V.22bis/V.22/V.21
エラー訂正プロトコル	CCITT V.42，MNPクラス５
漢字コード	シフトJIS

＊移転のため電話番号が変わっておりますので，ご注意ください．

Programmer's Page

MONTHLY MAGAZINE FOR PROGRAMMERS

月刊プログラマーズ・ページ8月号

(7月18日発売)

付録：5″ 2HDフロッピーディスク付き!!

特集

1.だれでもできるWindowsプログラミング

2.MS-C/C++,BC++クラスライブラリ活用法

VisualBasic，WindowsMAKERなどの優れたツールを使いこなして，容易に本格的なWindowsプログラムを作成する方法を紹介する．また，ObjectWindows，Microsoft Foundation Classesといったクラスライブラリを活用したWindowsプログラミングの方法を探る．

月刊プログラマーズ・ページ　1992年7月号
1992年7月1日発行
発行所　株式会社翔泳社
東京都港区西麻布2-22-1スリーアップ西麻布ビル
TEL 03-5467-0361　〒106
発行人　長廻健太郎
編集長　清水　隆
副編集長　鈴木嘉平
編集スタッフ　杉野貴美子＋芳山充弘
表紙デザイン　SE（翔泳社デザイン研究所）
広告スタッフ　相澤正夫＋澤本良知＋種村嘉彦
協力　根本裕三郎＋(株)ドモン・マインズ＋小林弘人（図版デザイン）
写植　株式会社東巧社＋株式会社トーヨー・プリント社
印刷　株式会社サンニチ印刷
定価　980円（本体951円，税29円）

EDITOR'S PAGE

●映画「無能の人」が評判になり，つげ義春が再浮上している．最初の作品集を神田の青林堂へ買いに行ったのは遠い昔のことだが，作品が文庫になって売れたり，「ねじ式」がパソコンゲームになったり，寡作にもかかわらず，時折世の中の注目を浴びてしまう作家であるようだ．映画，ゲームなどほかのメディアにも影響を及ぼすのは，作品の力によるものだろう．パソコンゲーム「ねじ式」は，つげ漫画のエピソードを集めて作られた不思議なアドベンチャーゲームで，つげ義春の作品を知る人には面白くプレイできるように仕上がっていた．デジタル８色でもきれいな絵を表示していたし，ソフトとしてのできは上々だったと思うのだが，商品としてはどうだったのだろうか．ちょっと気になるところだ．読者のみなさんは，ゲームの「ねじ式」をご存知でしたか？（TA）

●ボーランドのデベロッパーズコンファレンスから，なんとか生きて帰ってきました(笑)．コンファレンス自体はとても興味深く，楽しい時間を過ごせたのですが，困ったのが食事！　サラダに入っているブロッコリもカリフラワーも生のままでゴリゴリいうし，肉の付け合せのにんじんも一見煮てあるかのように見えながら，口に入れるとやっぱりゴリゴリ……．クッキーと林檎とポテトチップスの昼食なんてのもあったし．そんなときに，コンファレンスを抜け出して食べた中華料理のうまかったこと！ Iさん，ほんとうにありがとう．いやぁ～，やっぱりブロッコリはゆでなきゃだめよ．（嘉平）

●遅まきながら，ようやく話題の「バーコード・バトラー」を入手！　家中にあるバーコードをかき集めて，「これは強い！」「これはたいしたことない」などと，連休中はずっとはまっておりました．でも，もうすでに史上最強バーコードや，生命力，攻撃力などのポイントも下６桁で決まるという，種あかしが載った〝攻略本〟が出ていて，純粋にゲームとして楽しみたいワタシには，なんとなく興ざめです．でも，「ポン・ジュース vs 生協のジュース」や「サランラップ vs クレラップ」なんかの対戦は結構楽しめます．（ま）

●「20歳になったのが遠い昔のような気がする」．数年後に30歳を迎える僕が言ったのではなく，先日21歳になった子が僕に対して言った台詞だ．自分の歳を実感する瞬間である，しかも自分の誕生日でなく人の誕生日に…．(お詫び：今月の買物は額が大きかったのですが，諸般の事情より「今月の衝動買い」はお休みさせていただきます（←冗談）（yoshi）

愛読者カードご記入のお願い

本誌をお買い上げいただき、誠にありがとうございます。
お手数ですが、今後の誌面づくりの参考にさせていただきたいと存じますので、
各欄にご記入の上、ご返送くださるようお願い申しあげます。

月刊プログラマーズ・ページ READER'S SURVEY CARD 愛読者カード

本誌の号数　年　月号

本誌を知った理由
□1.書店店頭　□2.他社出版物(　)
□3.小社出版物　□4. DM　□5.口コミ

パソコンの所有形態(機種名)
□1.個人所有　(　)
□2.職場にある(　)
□3.持っていない

本誌を購入した書店　市(区)　書店

よく使用するソフトウェア

よく使用するOS

今月号でおもしろかった記事(いくつでも)

今後取り上げてもらいたい特集のテーマ

貴方が選ぶベスト3
パソコン雑誌 ①　②　③
ソフトウェア ①　②　③
パソコン本体 ①　②　③

本誌についてのご感想、お気づきの点、その他どのようなことでも結構ですので、ご意見をお聞かせください(「ドラゴン・ポスト＝読者の声」欄に掲載させていただく場合もあります。ペンネームをご希望の方はあわせてご記入ください)。

月刊プログラマーズ・ページ 広告資料請求カード

8	会社(学校)名		9	所属(学部)		10	従業員数	

※下記の該当する番号をご記入ください。

11	業種		12	職種		13	パソコン利用状況	

業種　①製造業　②金融・保険・不動産業　③商社・卸・小売業
④建設業　⑤通信・運輸　⑥システム・ソフト
⑦その他サービス業　⑧公務員　⑨その他

職種　①コンピュータ(ハード)技術者　②コンピュータ(ソフト)技術者　③エレクトロニクス技術者
④プログラマ　⑤その他の技術者　⑥専門職
⑦事務・営業職　⑧経営管理職　⑨商工サービス
⑩教職　⑪学生　⑫その他

パソコン利用状況　①会社で利用　②個人で利用　③会社でも個人でも利用　④利用していない

●希望される広告資料　年　月号

希望項目 ＼ 広告資料請求No.												
A 購入希望	A	A	A	A	A	A	A	A	A	A	A	A
B 説明希望	B	B	B	B	B	B	B	B	B	B	B	B
C 資料希望	C	C	C	C	C	C	C	C	C	C	C	C

定期購読のご案内

綴じ込みの振替用紙を使って、本誌の定期購読をお申し込みになれます。
1年間12冊＝11,760円（本体11,417円/税343円）
郵送料サービス

郵便はがき

106-00

料金受取人払
麻布局承認
905
差出有効期間
平成6年4月
30日まで
切手不要

東京都港区西麻布2-22-1
スリーアップ西麻布ビル
株式会社 翔泳社
マーケティング部 行

月刊プログラマーズ・ページ READER'S SURVEY CARD 愛読者カード

氏名	性別 □1男 □2女	生年月日 19 年 月 日
住所 〒		電話 （ ） －

職業
□1ソフト技術者 □2ハード技術者 □3技術系会社員
□4事務系会社員 □5営業系会社員 □6教師
□7一般公務員 □8経営管理職 □9理系学生
□10文系学生 □11その他（ ）

ユーザー区分
□1 プロ
□2 マニア
□3 一般ユーザー

プレゼント記入欄	クイズ・ランダムスクリーン回答欄
希望商品番号	答

郵便はがき

106-00

料金受取人払
麻布局承認
906
差出有効期間
平成6年4月
30日まで
切手不要

東京都港区西麻布2-22-1
スリーアップ西麻布ビル
株式会社 翔泳社
月刊プログラマーズ・ページ
広告資料請求係行

＊当社で記入

1	SE	2	受付No.	

月刊プログラマーズ・ページ広告資料請求カード

3	会員番号	

＊すでに会員番号をお持ちの方はご記入ください。

4	フリガナ 氏名	姓 名	5	生年月日	19 年 月 日

6	自宅住所	〒 電話 （ ）
7	会社住所	〒 電話 （ ）

資料送付希望先の番号に○印をつけてください。

払込通知票

通常払込料金加入者負担

口座番号	東京 4 - 15493
金額	11760
加入者名	株式会社 翔泳社
料金	払込み / 特殊 円
備考	
払込人住所氏名	※(郵便番号) TEL( )
受付局日附印	

各票の※印欄は、払込人において記載してください。

記載事項を訂正した場合は、その箇所に訂正印を押してください。

この払込通知票は、機械で使用しますので、下部の欄を汚さないよう特に御注意ください。また、本票を折り曲げたりしないでください。（郵政省）

切り取らないで郵便局にお出しください。

払込票

通常払込料金加入者負担

口座番号	東京 4 - 15493
加入者名	株式会社 翔泳社
金額	11760
払込人住所氏名	※
備考	
受付局日附印	

ここでは、DOS/Vの基本的な知識とともに、それがもたらすコンピューティングの世界を紹介しよう。

定期購読のご案内

- ■定期購読は、1年間12冊と限定させていただきます。
- ■購読料のお支払は、この振替用紙をご利用ください。銀行振込、現金書留によるご送金は、ご遠慮ください。
- ■振替用紙に、ご住所、お名前をご記入のうえ、郵便局の貯金窓口にてお支払ください。
- ■小社より領収書の発行はいたしませんので、郵便局にて「郵便振替払込金受領証」をお受取ください。
- ■お申し込みいただいた方には、弊社より直接発送いたしますので、到着が発売日より遅れることがございますが、ご了承ください。

この欄は、加入者あての通信にお使いください。

月刊プログラマーズ・ページ　定期購読直接申し込み書

Programmer's Page MONTHLY MAGAZINE FOR PROGRAMMERS

年　　月号より

年間購読料 11,760 円（税・送料込み）

■ご希望の購読開始月号をご記入ください。

■送付先（住所・氏名は正確にご記入ください）

住所	□□□-□□	フリガナ	
		氏名	
		TEL	

この払込通知票は、機械で使用しますので、下部の欄を汚さないよう特に御注意ください。また、本票を折り曲げたりしないでください。（郵政省）

郵便はがき

1 0 6 - 0 0

東京都港区西麻布2-22-1
スリーアップ西麻布ビル

株式会社 翔泳社
月刊プログラマーズ・ページ
広告資料請求係行

2	受付No.	

マーズ・ページ広告資料請求カード

＊すでに会員番号をお持ちの方はご記入ください。

名

5	生年月日	19　年　月　日

綴じ込みの
1

□7一般公務員　□8経営管理職　□9理系学生　□3一般ユーザー
□10文系学生　□11その他（　　　）

プレゼント記入欄	クイズ・ランダムスクリーン回答欄
希望商品番号	答

6	自宅住所	〒　電話（　　）	資料送付希望先の番号に○印をつけてください。
7	会社住所	〒　電話（　　）	

いま DOS/V の世界へ

いまDOS/Vが注目を浴びている．IBM PCとPS/2に，基本的にはハードウェアを追加することなく日本語環境を実現し，同時に英語環境でのソフトウェアの動作も実現するDOS/Vシステムは，価格的にも，機能的にもパーソナルコンピュータユーザーにとって十分に魅力的だ．とくに，MS-Windowsが普及しつつある状況において，IBM PCの特色であるフレキシブルなビデオシステムは威力を発揮する．また，プログラム開発者にとっては，米国などで発売された最新の処理系を利用できる環境を持つことができる．

ここでは，DOS/Vの基本的な知識とともに，それがもたらすコンピューティングの世界を紹介しよう．

DOS/Vとは何か？

田中　亘

DOS/Vという言葉が，パソコンの専門誌から一般雑誌のハイテク特集でも，ちらほらと目に付くようになってきた．おそらく，今のパソコンマニアのほとんどが，このDOS/Vなるものに注目しているはずだ．

しかし，用語が先行している割りには，まだDOS/Vの基本原理と，魅力が正しく伝わっていない．そこで，改めてDOS/Vの基礎知識とそれを取り巻く状況を解説しよう．

DOS/V 誕生の背景

DOS/Vというのは，日本IBMが自社製のPS/55シリーズ用に新しく開発したDOSだ．IBMではMS-DOSとはいわず，IBM DOSというがベースは同じMS-DOSだ．正式には，IBM DOS バージョン J5.0/Vという（DOS/Vが発表された当時は，バージョン4.0だった）．

DOS/V開発以前から，日本IBMは独自の日本語DOSを開発していたが，この独自DOSは，24ドットフォント表示を実現するために高解像度表示を必要としていたため，米国IBM PCやPS/2シリーズとの互換性を確保できないものだった．

むしろ，本家日本IBMよりも，国内AXメーカーの提供するMS-DOSとそのハードウェアの方が，米国IBM PCとの高い互換性を確立していた．

そんな背景を意識して，日本IBMでも，米国IBM PCとの互換性を確保しつつ，日本語処理のできるDOSを開発する必要性が高まってきた．その結果誕生したのがDOS/Vだった．

米国製PCとの互換性を意識した結果，いままでハードウェア能力に頼っていた日本語処理機能をすべてソフトウェアでまかなってしまおうという発想が生まれた．

その基本的な動作原理は，Macintoshの漢字Talkと同じ考え方なのだが，あちらはまったく設計思想の異なるパソコンなので，MS-DOSユーザーには，ピンとこないかもしれない．

PC-9801などの国産パソコンは，IBM PCと同様にMS-DOSを基本ソフトとするパソコンだが，ハードウェアの基本構造が異なるために，米国PCとの互換性はない．そこで，米国PCとハードウェアの基本構造を同じにした日本語パソコンを作ろうということで，できたのがAXパソコンだ．

AXパソコンも，ハードウェアそのものは米国のIBM PCとの互換性を確立していたが，日本語表示の部分に限っては，漢字ROMと同じ原理の専用拡張ボードを利用していた．

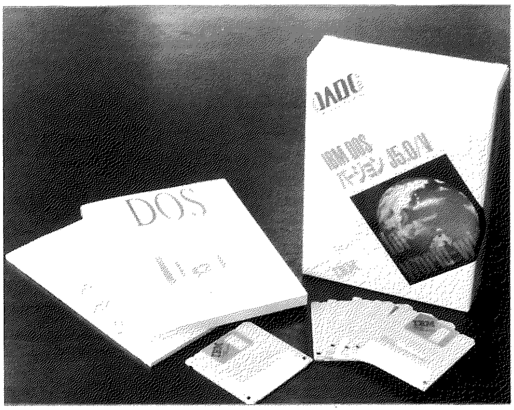
その理由は，初期のパソコンでは，日本語処理をソフトウェアだけでまかなえるだけの充分な処理能力がなかったからだ．漢字１文字の文字情報は，アルファベットの２倍あり，さらにその数となると，7000文字を越える数になる．これだけのデータを効率良く表示するためには，やはり専用のハードウェアが必要だった．

しかし，高性能な32ビットCPU80386の登場によって状況は好転した．ソフトウェア的に日本語を処理しても，その他アプリケーションソフトの性能を下げることなく，充分実用的な機能を提供できるようになったのだ．

その結果，DOS/Vは，漢字ROMなどの日本語表示のためのハードウェアをもたない米国製のパソコンでも，OSによって日本語を表示できるようにするDOSとなった．そのため，IBM

PC/AT互換機であれば，外国製のパソコンでも，日本語表示や日本語MS-Windows 3.0の利用を実現できるようになった．

IBM DOS 5.0/Vのパッケージこの春から出荷されたパッケージは，バージョン5.02になっている



OADG標準とされるA01キーボードこのほか，日本IBMの1型，2型，AX，J-3100のキーボードがOADGでサポートされる



DOS/Vマシンとは

DOS/Vを利用できるマシンとは，次のような条件を満たしているパソコンに限られる．

それは，IBM PC/ATかPS/2およびPS/55シリーズ，あるいはその互換機で，80286以上のCPUを搭載し，VGAというグラフィック表示能力を持ち，標準搭載メモリの他に，256KB以上の拡張メモリか漢字ROMを搭載した機種だ．

VGAとは，米国のPS/2で採用されているグラフィック表示の規格名称で，横640ドット，縦480ドットの解像度でカラー16色表示を実現する．また，横320ドット，縦200ドットのモードでは256色を表示できる．PC-9800シリーズのノーマルモードでは，縦640ドット横400ドットでカラーは16色だから，VGAの方が表示能力においては優れている．また，VGAに対して上位互換性をもっているスーパーVGAやXGAでもDOS/Vは動作する．

このハードウェア基本構成に加えて，効率良くDOS/Vを利用するためには，ハードディスクも必須周辺機器となる．

DOS/Vは，画面に表示するための日本語文字データをすべてデータファイルから読み出すため，フロッピーディスクだけではまかないきれないからだ．

これらの条件を整えたハードウェアであれば，基本的にはDOS/V利用条件を満たすが，さらに注意するならば，キーボードも重要だ．

外国製のパソコンには漢字変換操作のためのキーが用意されていない．そのため，単にDOS/Vを利用しても，肝心の日本語が入力できなくなるケースも多い．

米国製PCとPS/55シリーズのキーボードを並べて比べてみるとよくわかるが，キー配列やキートップの名称が違う．外国製パソコンの利用に精通しているユーザーならば別だが，入門者にとっては非常に困惑する問題だ．

DOS/Vに興味があっても，パソコンの利用経験が浅い場合は，マニアのように外国製の安価なPC互換機に飛びつくと，実用にならないこともある．注意が必要だ．

入門者がDOS/Vに手を出す場合は，日本国内でもしっかりしたサポートをしてくれるDOS/V対応機種を購入するべきだろう．

1台のパソコンを日英両用に

さて，以上の動作環境を整えたら，後はDOS/Vのインストールとなる．

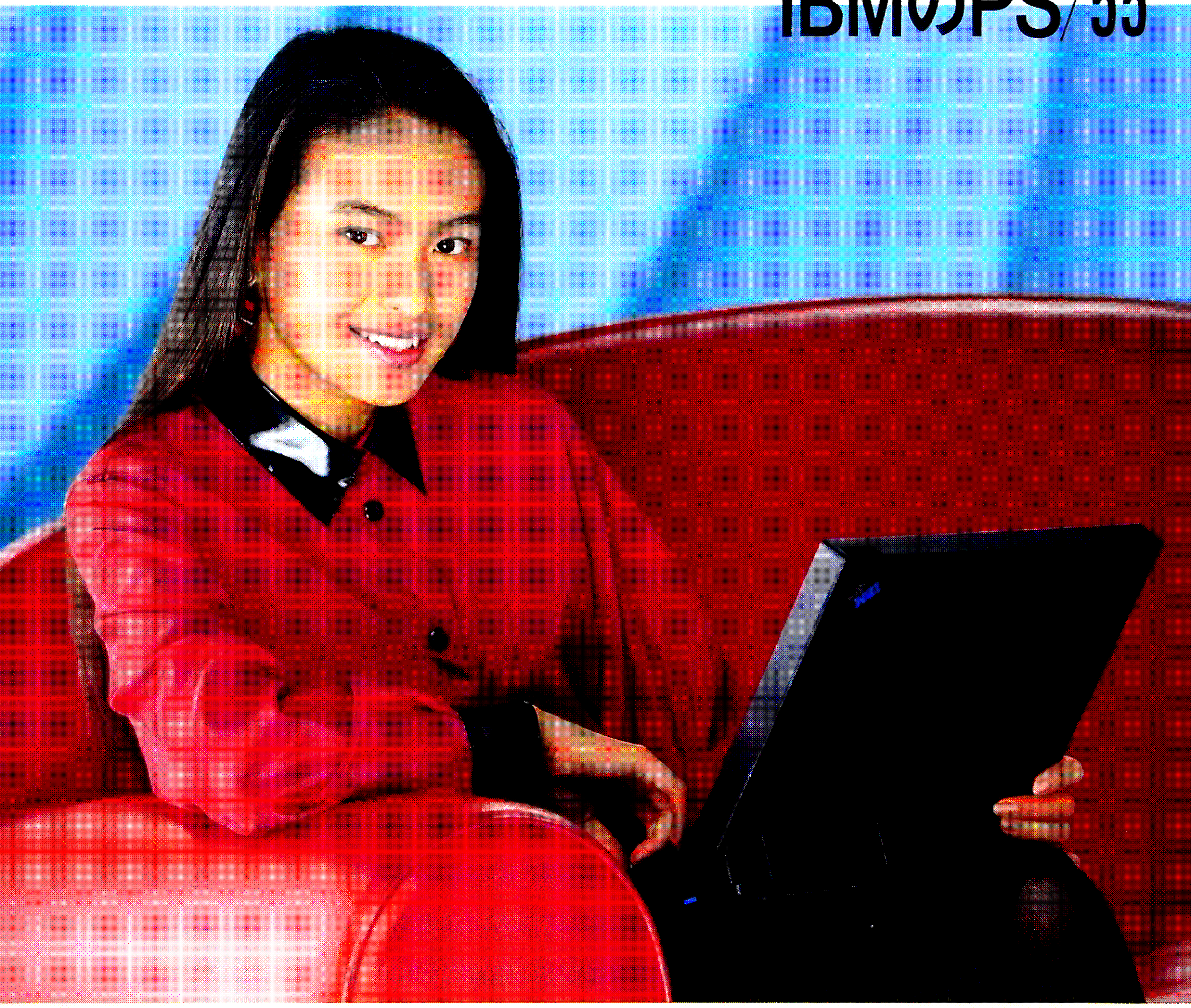
DOS/Vのインストールは，Ver.5.0から，MS-DOSの標準インストール処理の一部に組み入れられているので，画面との対話形式で，ハードディスクに導入できる．

導入を完了すると，config.sysにfont.sysとdisp.sysという2つのデバイスドライバが登録され，拡張子.fonというファイルがルートディレクトリにコピーされる．

また，countryという設定に日本語の場合は国番号が定義される．

丁度，日本語入力FEPが，デバイスドライバというソフトウェア技術だけで日本語入力を実現しているように，DOS/

DOS/V、XGA/Windows
その実力を確かめてください。
IBMのPS/55



世界中の豊富なソフトがつかえる話題のDOS／Vパソコン。今、日本のパソコン事情を大きく変えようとしています。IBMのPS／55*は、DOS／Vパソコンの実力機です。簡単でスピーディなパソコン操作で人気のPS／55 note、きわめて精緻なグラフィック画像を実現したXGA*／Windows**対応のPS／55Z　30Uなど。これからの主流となるにふさわしい卓越したパフォーマンスが魅力です。時代が求めるパソコン、それがPS／55



Windowsが走る
ジャストA4サイズのノートパソコン

PS/55 note



XGAを採用した
パーソナル・デスクトップの最高峰

PS/55Z 30U

OADC
PC Open Architecture Developers' Group

PS/55note、PS/55Z、Windows3.0のお問い合わせ、資料請求は下記までどうぞ。

PSサポートデスク0120-055302

日本アイ・ビー・エム株式会社

*IBM、PS/55、XGAは、IBM Corp.(米国)の商標。**Windowsは、米国マイクロソフト社の商標。

〈資料請求番号003〉



パーソナルコンピュータ（日本語DOS/V＆IBM-PC/AT互換機）

48AG-STD	標準価格 598,000円（税別）	48AG-H120	標準価格 798,000円（税別）
38HM-STD	標準価格 448,000円（税別）	38HM-H120	標準価格 648,000円（税別）
38FV-STD	標準価格 268,000円（税別）	38FV-H40	標準価格 348,000円（税別）

DOS/V 48AG

DOS/V & ATマシン：新設計のコンパクトサイズ日本語DOS/V専用106キーボードと3.5"ドライブを搭載した高性能DOS/Vマシンです。また、MS-DOSやPC-DOSはもちろん新OSとして注目のIBM OS/2 V2.0やDR-DOSマシンとしても高性能を発揮します。日本IBMの日本語Windows 3.Xのハイレゾ・ドライバー＊1や各種英文アプリケーション用グラフィックドライバー標準添付で高解像度SVGAの実力を発揮できます。

バイリンガル・マシンとして、ATアプリケーションとの互換性を充分に発揮するために、5"ドライブも搭載することで、膨大なATアプリケーションに対応しています。バスマウスも標準装備しております。

ハイコスト・パフォーマンス・マシン群：48AGは486DX-33MHzで駆動し、内部8KB、外部256KBのキャッシュメモリ搭載と強力な演算機能によって、広範囲の分野で超高速動作を実現します。

38HMは386DX-40MHzで駆動し、64KBのキャッシュメモリ搭載によって、WindowsやOS/2のビジュアルシェルで最大の能力を発揮します。

38FVは386SX-25MHzで駆動し、各種アプリケーションを堅実にこなすハイコスト・パフォーマンス・マシンです。

重装備パソコン：標準でDOS/V用ソフト用の3.5"ドライブとATアプリケーション用の5"ドライブ両方を搭載。グラフィック機能もVGA上位互換のSVGAを搭載し最大解像度は1024×768ドット（262,144色中256色同時発色可能）です。また、標準メモリ4MB、ATバス装備はもちろん、シリアルポート×2、プリンタポートを装備し、ハードディスクも内蔵可能（最大2基内蔵可能）です。

○DOS/V、日本語Windows、CRTはオプションです。
○MHz表示は、CPU処理速度の規定値を表示させています。
○広告で使用のCRTはJUM-1482AN（標準価格128,000円）です。
＊1　日本語Windows用ディスプレイドライバーは、1024×768ドット（256/16色）、800×600ドット（256/16色）、640×480ドット（256色）です。

○これらのパソコンは全て日本語DOS/V仕様です。
○記載されている商品の価格には消費税は含まれておりません。御購入の際、消費税が付加されます。
○御求めは、全国有名パソコンショップでどうぞ。

PLUS ONE PERSONAL ENVIRONMENT & TECHNOLOGY

関西電機株式会社プラスワン
〒101　東京都千代田区岩本町3－2－10 SN岩本町ビル
プラスワン問い合せ：TEL.03-5687-6912
テクニカルサポート：TEL.03-5687-6913
プラスワン　つくば：TEL.0298-56-8851

〈資料請求番号004〉

Vは，font. sysとdisp. sysという2つのドライバソフトウェアによって，DOS上で日本語を表示できるようにしている．

英語と日本語の切り替えは，chev. exeというユーティリティで切り替える．

英語モードにすると，外国製のアプリケーションやゲームを起動できる．そして，日本語モードにすれば，DOS/V対応の日本語アプリケーションや，日本語MS-Windows 3.0を起動可能だ．

DOS/Vを利用すると，1台のパソコンを日英両用にできる魅力が得られる．また，利用するパソコンが，米国標準といわれるIBM PC互換機なので，ビデオ出力ボードのような，米国製の高機能な周辺機器をリアルタイムで利用できるようになる．

もっとも，なにもかも最先端のパソコンと周辺機器で揃えようとすると，そのサポート先も情報もすべて英語になってしまうので，国内のビジネス利用には，まだ普及していない．

現在は，プログラマや一部のホビーユーザーを中心に，互換機利用が流行している．

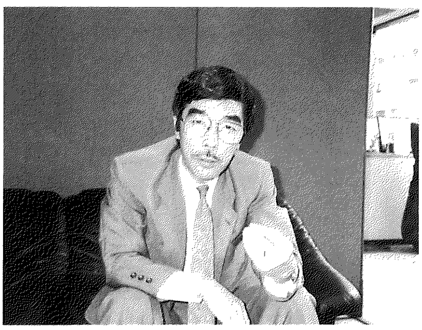
日本のパソコンメーカーも，日本IBMの提唱するDOS/V仕様に賛同し，OADGという団体を中心に，DOS/Vが動作するパソコンを開発している．

やがては，国内のOADG加盟メーカーを中心としたDOS/Vマシンが，手軽に入手できるようになるはずだ．

IBM DOS 5.0/Vのパッケージこの春から出荷されたパッケージは，バージョン5.02になっている．

OADG標準とされるA01キーボードこのほか，日本IBMの1型，2型，AX，J-3100のキーボードがOADGでサポートされる．

FORUM
関西電機株式会社



亀山武暉・常務取締役

PI（プラスワン）シリーズでDOS/Vマーケットに登場した関西電機．386または486CPUとSVGAを搭載した高機能パソコンが売りものだ．国内の会社としてはDOS/Vへの対応は早い時期から行っており，ユーザーからの注目度も高い．

「考えていた以上にDOS/Vマシンへの流れは速い」と，話すのは，同社の常務取締役・亀山武暉氏だ．「うちは周辺機器が中心の会社ですが，海外のパソコンの価格に刺激されてPIシリーズを出しました．大手メーカーが参入する前にDOS/Vマシンを投入することで，市場の下地を作ることができればと考えていたのです．大手が参入すれば，うちとしては本来の周辺機器で仕事ができると……．それが予想以上に早く，さまざまなところからマシンが出てくるようになっています」(同氏)．DOS/Vマシンの信頼性を高めるために，長い目でみた展開を考えていきたいという．

DOS/Vのプログラミング環境

柳沢健二

DOS/Vマシンを選ぶ理由

“世界標準”を旗印に，海外パソコン文化の香りを漂わせながら登場してきた「DOS/V」も，登場以来はや1年半．このごろではIBM PC/AT互換アーキテクチャのパソコンを日本のパソコンユーザーに「検討に値する次期購入

マシン候補の一つ」とイメージづけることに成功しつつあるようだ．結果として既存の国産パソコンを選ぶにしてもマッキントッシュを選ぶにしても，比較対象としてはDOS/Vをある程度意識している人が，今ではかなり多いことだろう．

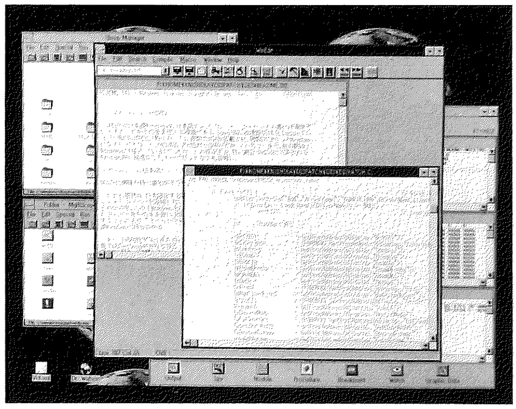
もっとも実際の普及台数はまだ多くないらしいが，パワーユーザーやパソコン界のオピニオン・リーダーとでもいうべき人たちが，DOS/Vに乗り換える例が増えてきていることは，筆者の実感として事実である．

たとえばあるDOS汎用フリーソフトウェア作者として知られる北海道在住のY氏は，今春，通信販売でAT互換機（80486DX／33MHz）を購入したという．購入動機は「コンパイルが速いマシンがほしかったから」とのことだ．“DOSマシンである／安い／日本語が使える／速い”．DOS汎用ソフト開発が中心ならこの条件を満たせば充分だから，確かに「486マシンが安い」DOS/Vは魅力的だろう．

一方，京都府在住で，これまた有名なファイル管理ツール（やはりフリーソフトウェア）の作者であるI氏は，かつて愛用していたPC98は今ではプログラムの動作テストに使うだけで，メインはすっかりDOS/Vだそうだ．プログラマの視点から見てDOS/Vの方が「作って面白く，動かして楽しく，使って便利」と感じた，というのが移行の主な理由のようだ．実際，I氏のツールはPC98版とIBM版（DOS/V版）が公開されて市販ソフト顔負けの人気を集めているが，ハードウェアの相違をそのまま反映して，前景・背景各16色中から自由に指定可能な画面色カスタマイズ機能など，細部ではIBM版の方が機能豊富だ．



ハイテキストの表示画面　SVGAの高解像度をテキスト画面でも活用できるハイテキストドライバ



高解像度Windows1280×1024ドットの高解像度Windows．仕様書とマニュアルと同時に表示しながらプログラムを書くことも可能だ

たちはだかるハードル

このI氏の場合，オリジナルの自作ツールをスムーズにDOS/Vに移植できたことが，DOS/Vへの本格移行のポイントということになる．本誌読者の中にもI氏と同様，特定機種依存の自作ツールをいろいろお持ちの方も多いことだろう．ということは，それらを自力で移植するか，同等の機能を持ったDOS/V対応ソフトを見つけない限り，DOS/Vへの移行はむずかしいことになる．

「ワープロソフト一本あればOK」という人ならともかく，手持ちのパソコンと深くつきあってきたプログラマの場合は，「プログラミング環境の再整備」「IBM PC系ハードウェアやDOS/Vの機能の勉強」「手持ちツール類の移植作業」……などなど，やらねばならないことはけっこう多く，それができたときDOS/Vへの移行の意義も出てくるわけだ．そういう意味からは，筆者としてもあまり気安くDOS/Vをおすすめするわけにはいかない．うっかり入り込んだら後悔するハメになるかもしれない…….

ハードウェア

……と，そんな筆者のおどしにもめげず？あなたがDOS/Vマシン購入を決意した場合について，少し具体的に「DOS/V上のプログラミング環境」での重要なポイントになりそうな事項について考えてみよう．

まずはDOS/Vが動作するパソコン本体．パワフルなCPUを搭載したマシンがわりあい気軽に買えてしまうのが

IBM PC/AT 互換機
全米で第1位の実績

PACKARD BELL

America grew up listening to us. It still does.



PB486DX-33(ISA/バス)

CPU：80486DX-33●メモリ：4MB(最大32MB)●キャッシュメモリ：オプション(256KB迄)●FDD：5.25″1.2MB×1, 3.5″1.44MB×1●インターフェイス：102Keyboard シリアル×2, パラレル×1　PS/2マウスポート付 S-VGAモニターポート付(モデムポート付)●拡張スロット：16bit×4●標準ソフト：MS-DOS Ver.5.0英文マニュアル付

希望小売価格¥370,000のところを
特別提供価格
¥298,000

■システム価格例

PB486DX-33	¥298,000
200MB HDD	¥138,000
14″ S-VGAモニター	
+ (1024×768)	¥ 65,000
合計	¥501,000

PB486SX-20

CPU：80486SX-20●メモリ：4MB(最大32MB)●キャッシュメモリ：オプション(256KB迄)●FDD：5.25″1.2MB×1, 3.5″1.44MB×1●インターフェイス：102Keyboard シリアル×2, パラレル×1　PS/2マウスポート付 S-VGAモニターポート付(モデムポート付)●拡張スロット：16bit×4●標準ソフト：MS-DOS Ver.5.0英文マニュアル付

希望小売価格¥320,000のところを
特別提供価格
¥258,000

■システム価格例

PB486SX-20	¥258,000
100MB HDD	¥ 78,000
14″ S-VGAモニター	
+ (1024×768)	¥ 65,000
合計	¥401,000

PB55(SX20)

CPU：80386SX-20●メモリ：2MB(最大16MB)●キャッシュメモリ：16KB(最大32KB)●FDD：5.25″1.2MB×1, 3.5″1.44MB×1●インターフェイス：102Keyboard シリアル×2, パラレル×1　PS/2マウスポート付 S-VGAモニターポート付(モデムポート付)●拡張スロット：16bit×4●標準ソフト：MS-DOS Ver.5.0英文マニュアル付

希望小売価格¥210,000のところを
特別提供価格
¥168,000

■システム価格例

PB55(SX20)	¥168,000
40MB HDD	¥ 39,000
14″ S-VGAモニター	
+ (1024×768)	¥ 65,000
合計	¥272,000

PB8517SV

17インチ・マルチスキャン Super VGAモニター (1280×1024)

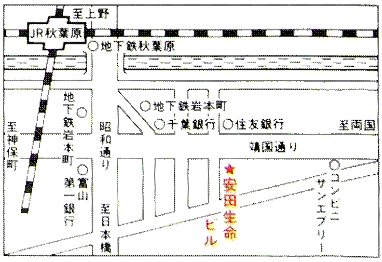
希望小売価格¥198,000のところを
特別提供価格 **¥158,000**

平日9：00〜6：00
土曜9：00〜3：00
日祭休み

全製品、日本で厳密なる検査の上出荷しています。
全製品即納体制です!!

PACKARD BELL 販売代理店 CD 千葉電子株式会社 会社名、商品名は各社の商標または登録商標です。

お問合せは……
PBショールーム 0120 **0120-009035**
千代田区岩本町3-5-5 安田生命岩本町ビル8階 ☎(03)5820-0701(代)
FAX.(03)5820-0710



物流サービスセンター　〒281 千葉市山王町382-2 FAX.(0434)24-2412

〈資料請求番号005〉

HEWLETT PACKARD

0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0



独自のサーマルインクジェット方式。

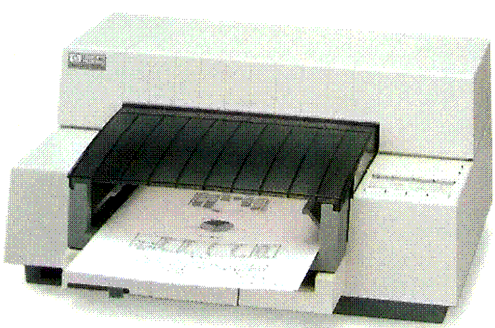
HPが1984年に世界で初めて発売した、熱によりインクを噴射する技術「サーマルインクジェット」方式プリンタ。この実用化を可能にしたのは、インクを噴射するノズルに用いられた高精度の薄膜加工技術です。そしてインクの噴射は、1秒間にじつに5000回。静かで美しいうえに、「とにかく速い」秘密はここにあります。

この価格で、レーザプリンタに迫る。

まず「速さ」。漢字を高品位モードで110字/秒、高速モードにすれば160字/秒。10万円以下では類をみないこの速さが、仕事の能率をグーンとアップさせます。そして44dBの「静かさ」。となりで電話をしていても、夜中でも気兼ねなくプリントできます。さらに300dpiの「美しさ」も魅力。文字もグラフィクスも高品位で印字します。

身近に使いやすい。末永く使いやすい。

「使いやすさ」も、デクジェット500Jの大きな特長です。一度に100枚の用紙をセットできるカットシートフィーダを標準装備。紙の扱いが楽なフロントローディング（前面給紙方式）を採用。そして3年間の長期保証付き。世界中で300万台以上のインクジェットプリンタの販売実績を持つ、HPならではの自信の証しです。



日本語インクジェットプリンタ

HP DeskJet 500J

標準小売価格：98,000円（税別）

製品番号 C2603A

●対応機種
NEC PC-98シリーズ、EPSON・PCシリーズ、東芝J-3100シリーズ、IBM・PC-XT/AT、各社AX、DOS/V仕様パソコン

●別売アクセサリー

インクカートリッジ	51608A	3,200円
大容量インクカートリッジ	51626A	5,200円
丸ゴシックフォントカートリッジ	C3000A	9,800円

お問い合せはフリーダイヤルで
カストマ・コミュニケーションセンタ
0120-081444

横河・ヒューレット・パッカード株式会社
本社 〒168 東京都杉並区高井戸東3-29-21
CPOIP02 AD-2043

〈資料請求番号006〉

DOS/Vマシンの魅力の一つであることはY氏の例に見られる通りだが，もう一つ，「ディスプレイはなるべく高解像度対応で，サイズもある程度大きなものが望ましい」という点をおさえておきたい．

VGA搭載のPCでは，英語環境で80×50行モードが“標準で”サポートされており，プログラミング言語の統合環境等では当然このモードが利用できる．また，各社のSVGAディスプレイボードの機能を利用して，日本語環境でも100×37行あるいはそれ以上の表示を可能にする「ハイテキストドライバ」がフリーソフトウェアとして流通しており（もっと高解像度な版もあり，こちらは有償），「VZ」「WXII+」「知子の情報」などといった市販ソフトがこのモードをサポートしている．

画面の見通しが良くなれば，それだけプログラミング作業もはかどるというもの．それには当然ディスプレイも大きいほうが望ましいわけだ．特にこのところPCの画面回りの進化は顕著で，1280×1024ドット256色SVGAボード，なんていうのも案外安価に手に入るが，手持ちのディスプレイの解像度が追随できなければ意味がない．

プログラミング言語

実は，正式に「DOS/V対応」として現在市販されているプログラミング言語は，日本IBM純正でOS/2にも対応の10万円以上するコンパイラ群（C，PASCAL，BASIC，FORTRAN），それにマイクロソフト製のMS-C6.0……と，どれも高価で大がかりなものばかりだ．経済的で軽快なDOS/V対応プログラミング言語みたいなものは，残念ながら「一つも存在しない」．

では，たとえばTurboC(++)とかTurboPascal，あるいはQuick Basicは使えないかというと，今書いたのはどれも元来IBM用がオリジナルだから，当然DOS/Vでも「英語版なら」使える．

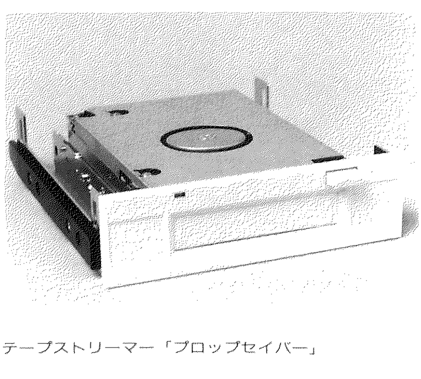
この場合，いわゆる「統合環境」は英語環境でしか使用できないが，コマンドライン版のコンパイラはむろん日本語環境から使用可能で，画面関係やファイル，文字の入出力部分にさえ気を使えば，できあがったプログラムも日本語環境で動作する．

「画面関係に気を使う」とはどういうことかというと，DOS/Vの日本語テキスト画面で文字を表示させるには必ずBIOSの機能を呼び出す必要があるのに対し，英語版に付属してくる画面関係のライブラリはVRAMに直接アクセスして表示を高速化しているのが普通で，そのままでは日本語環境では文字が表示されないのだ．このため日本語対応プログラムを開発するためには，次のどれかの工夫が必要となる．

1) DOS標準出力を使用し，カーソル制御等はエスケープ・シークエンスに頼る．
2) 利用したいライブラリに「画面制御はBIOS経由で行う」オプションが存在するならそれを利用する．
3) パッチをあてるかソースを修正して，ライブラリを日本語環境対応に改造する．
4) 自前でライブラリを作り直す．

……面倒といえば面倒だし，日英どちらの環境でも同じ感覚で利用できる安価な「DOS/V対応版」が早く出てることを願わずにいられないが，TCやTPなど，BIOS/DOS割り込みルーチンが簡単に記述できるタイプの言語な

FORUM ビルトイン


テープストリーマー「プロップセイバー」

ビルトインは，株式会社クラスターが経営するDOS/Vショップだ．台湾パソコンメーカーEFA（イーファ）と契約しており，店頭のパソコンもEFA製である．パソコンのケース，ハードディスク，フロッピードライブなど，コンポーネントで購入できるのもマニアにとって魅力だ．また，バックアップテープストリーマーを発売するなど自社製品の開発にも熱心だ．
「Windowsが普及してきている中でデータも大容量化しています．200MBのハードディスクのバックアップをとるにしても，フロッピーでは容易なことではありません．今回発売するテープストリーマーは，そうした状況の中で活躍できると思います．この製品は，5インチのベイに入れて接続するだけですから，SCSIカードを購入する必要もありません．また，当社で開発した製品ですから，国内でサポートできるということで安心して使っていただけると思います」と，同店の田中店長．製品名は「プロップセイバー」といい，価格は64,800円．ストリーマーがぐんと身近になった．

ら，どのみち致命的というほど不便ではないというのが実感だ．

エディタ

統合環境は日本語環境で使えないから，好むと好まざるとにかかわらず，日本語環境でのソースコード入力編集には別にエディタが必要となる．この面ではVZ, Mifes, Briefなどがすでに DOS/Vに対応しているため障害はなく，特にVZがすでに「ハイテキスト」環境に対応しているのは先に触れた通りだ．

その他のユーティリティ

IBM PCの世界は文句なしの“プログラミング先進環境”だから，言語だけではなく，さまざまなユーティリティ，周辺ツール類，資料類もオリジナル版がいちはやく入手でき，そのまま活用可能だ．高機能なデバッガ，ソースコードジェネレータ，DOS非公開ファンクションのハイパーテキスト・データベース……その他もろもろ，実にいろんなものが出回っている．それらのうち国産パソコン向けに移植されるのはごく一部分にすぎないし，移植にかかるタイムラグもある．しかしDOS/Vにはその制限はない．

というわけで，はなはだ急ぎ足の原稿で申し訳ないが，“英語版でよければ常に世界の最先端”……DOS/Vが現在提供できるこのプログラミング環境にどれだけ魅力を感じるか，感じないか．その判断は皆さんにおまかせすることにしよう．

DOS/V の 基礎知識

IBM PC

1981年に発表されたIBM PCは，1983年にハードディスクに対応したPC/XT，1985年に80286を搭載し拡張バスを強化したPC/ATへと進化した．米国などで販売されているPC互換機はPC/ATを基準としたものだが，CPUは386や486が主流になっている．日本以外の世界のMS-DOSパソコンは，ほとんどPC互換機で占められている．

PS/2

IBMがPC/ATの後継として発表したモデル．バスアーキテクチャを変更したために，PC/ATとは拡張ボードなどハードウェアの互換性はないが，ソフトウェアはほとんどのものが共通で動作する．現在，PCの標準的なビデオシステムとなっているVGAは，PS/2のビデオシステムとして登場したもの．DOS/Vパソコンには，PC/AT互換のもの（日本IBMの5510Sや多くのPC互換機）とPS/2互換のもの（PS/55シリーズや三菱電機のアプリコット）がある．

DOS

初代IBM PCとともに世に出たPC-DOSは，マイクロソフト社からもMS-DOSの名前で供給された．以来同じバージョンのDOSがIBM，マイクロソフトからOEMを受けたメーカー，マイクロソフト社などから供給されることになった．

PC WINDOWS/TOKYO
EXPOSITION & CONFERENCE
7月7日(火)～7月9日(木)
AM10：00 ～ PM5：00
幕張メッセにてお待ちしております。
(ブース No.105)

クラスターから低価格機 ETシリーズ新発売!!
PS シリーズはパワーアップして、更にお求めやすくなりました。

オールニュープライス!

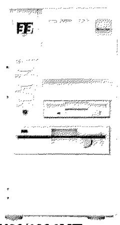
開発製造
EFA CORPORATION

DynaStar パソコンシリーズ 1年間保証

DOS/V

PS－386WX/33 大好評!
更にお安くなりました!

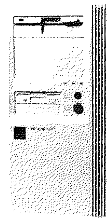
EFA社製34M33/50マザーボード採用マシン将来 CPU を乗せ換えるだけで 386 DXから486SX25、DX33 または DX 50にグレードアップできる画期的マザーボードを採用しています。



DynaStar PS－386WX33/1204MT
キャッシュ 128KB･HDD120MB

PS－486DX/33 噂のマシン
プロップセーバー付

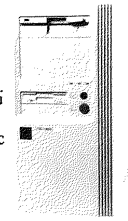
超高速HDD･IDE キャッシュコントローラを実装! CPU が早くても I/O 関係が遅ければ早くなりません。I/O バススピード最大12MHzで付いていないとの差は歴然! 雲泥の差です!



DynaStar PS－4806DX33/2108GTX
キャッシュ 256KB･HDD210MB

PS－486DX/50 超高速マシン
プロップセーバー付

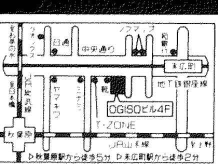
各専門誌で絶賛の超高速HDD･キャッシュコントローラを実装ディスクのシークタイムを最高0.5ms以下! データ転送速度 5MB/sec! RAM8MB でヒット率抜群です。



DynaStar PS－486DX50/2458GTX
キャッシュ 256KB･HDD245MB

	PS－386WX/33	PS－486DX/33	PS－486DX/50
CPU	INTEL386DX33MHz	INTEL486DX33MHz (冷却ファン付)	INTEL486DX50MHz (冷却ファン付)
キャッシュ RAM	128KB (MAX256KB)	256KB (MAX)	256KB (MAX)
メイン RAM	4MB (MAX32MB)	8MB (MAX64MB)	8MB (MAX64MB)
HDD	120MB	210MB	245MB
ストリーマ	オプション	CT－803 × 1	CT803 × 1
IDE キャッシュコントローラ	オプション	DC－600/4MB RAM 付	DC－600/4MB RAM 付
拡張スロット	AT バス × 7	AT バス × 5 ・ XT バス × 2	AT バス × 5 ・ XT バス × 2
電源	200W 50/60Hz 100 ～ 120V･220 ～ 240V		
共通仕様	BIOS :AMI HDD :3.5"(60MB ～各種) FDD :TEAC 製 3.5"(1.44MB)+5.25"(1.2MB) 計 2 台搭載 インターフェース : 2 シリアル、1 パラレル、1 ゲームポート キーボード : 日本語 106 キー・キーボード付		
S－VGA カード いろいろなカードを取り揃えております。※ ET シリーズは固定になります。※ⓑ・ⓒはオプションで金額が異なります。	ⓐ Tseng lab 純正 ET－4000MEGA VGA 1280 × 1024(16 色)、1024 × 768(256 色)、800 × 600(65535 色) ハイカラーモード付 1MBRAM 付 NEW DACIC 付 ⓑ ORCHID 社製 S3 チップ使用 Fahrenheit1280 Hi－COLOR モード付 1MBRAM 付 ※プラス ¥20,000UP ⓒ Diamond 社製 S3 チップ使用 STEALTH/VRAM Hi－COLOR モード付 1MBRAM 付 ※プラス ¥23,000UP ⓑ・ⓒ共に、1280 × 960(16 色)、1024 × 768(256 色)、640 × 480(32768 色)		
ケース (サイズ)	MT: ミニタワー (D415 × W200 × H345)	GT: グランドタワー (D414 × W220 × H440)	
	¥259,000	**¥515,000**	**¥589,000**

互換機専門ショップ

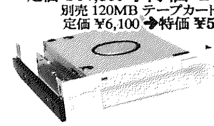


バックアップストリーマー

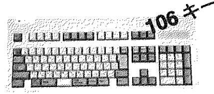
HDD も 100MB を越える時代になりましたが、バックアップはフロッピーですか? 実際 HDD がクラッシュした時のリストアは何十枚あって大変です。データと時間はお金では買えません。そこで低価格を実現した当社オリジナル「プロップセーバー」の登場! 国産品なのでダブルで安心です。プロップセーバーは空いている 5 インチベイに挿入しフロッピーケーブルと電源ケーブルをつなぐだけのビルトインタイプです。約 3MB/ 分と高速にあなたの作り上げた大切なデータを圧縮しながらバックアップします。120MB テープカートリッジ 1 本 (Qic80 フォーマット済) とバックアップソフト (DOS･WINDOWS 共に英語モード用) 付です。

プロップセーバー CT-803 120MBテープ1本付

定価 ¥64,800 ➔特価 **¥59,000**
別売 120MB テープカートリッジ
定価 ¥6,100 ➔特価 ¥5,700



日本語キーボード

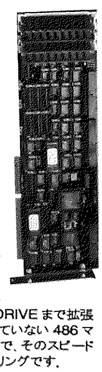


DOS/V WINDOWS 対応
JKB－02
クリックタイプ
メーカー直結価格 **¥11,800**

IDE キャッシュコントローラー

TEKRAM社日本総代理

マシン全体のパフォーマンスは、数字だけでは計れません。例えば、CPU が早くても I/O 関係が遅ければ結果的にマシン全体のパフォーマンスは早くなりません。これは、486 等の高速 CPU では更に顕著です。そこでクラスターとして出した結論が、このキャッシュつき IDE キャッシュコントローラです。最大 5MB/s の転送レート、HDD シークタイム最大 0.4ms 以下、最大 I/O バススピード 12 MHz の性能をもち、また、4 台まで IDE タイプの HDD を接続でき、最大 4 GB CAPACITY PER DRIVE まで拡張できます。実際これの付いていない 486 マシンと比べれば、一目瞭然で、そのスピードはまさに小気味良いフィーリングです。



テックラム DC－600
本体のみ **¥40,000**(RAM ナシ)
RAM 1MB SIMM付 **¥46,000**
RAM 4MB SIMM付 **¥65,000**
業販お問い合わせ下さい。

女性スタッフ大募集!!

クラスター直営パソコンショップ

Built in. Part 1

営業時間 －月・火・水・金－ PM12：00 ～ PM7：00
－ 土・日・祭 － AM11：00 ～ PM7：00
木曜定休
●しばらくの間第 2、第 3 水曜休ませていただきます。

TEL：03 (3257) 3264 千代田区外神田 4-4-3 OGISOビル4F

〈資料請求番号007〉

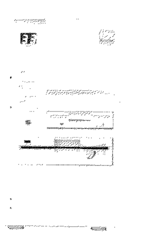
efa

私どもクラスターは高品質製品を低価格でお届けいたします。

ET−486DX/33−50 新発売!

ロープライス機の登場です。

しかしクラスターはスペックに妥協しません。こつこつとパワーアップしていくのも楽しみの一つです。
※この機種は仕様変更対象外です。



DynaStar ET−486DX33/1704MT
キャッシュ 256KB·HDD170MB
DynaStar ET−486DX50/1704MT
キャッシュ 256KB·HDD170MB

486DX/50 ローカルバス 3本搭載機 新登場!

7月発売予定 予約受付中

ローカルバス採用! プロップセーバー付

クラスターで大好評の IDE キャッシュコントローラ(DC-600)とビデオカード(ET-4000)ローカルバス版を搭載!アベレージシークタイム 0.3ms 以下!最大データトランスファーレート 20MB/s の驚異!

拡張スロットにはローカルバス 3 本、ISA バス 5 本を持っています。ローカルバス 3 本にはローカルバス専用 IDE キャッシュコントローラ (DC−660) と S−VGA カード (Tsenglab ET−4000 使用) の 2 本を使用し、もう 1 本は拡張を重視し将来のため (SCSIⅡ、リアルタイムビデオデジタイザー等) にご用意しました。当然、I/O バススピードはシステムロックと同じになります。また、このローカルバス 3 本はそのまま ISA バスとしてもご使用になれるよう設計されていますので、残りの 1 本を ISA バスとしての使用も可能です。

キャッシュ 256KB·HDD245MB

ET−486DX33/1704MT	ET−486DX50/1704MT	486DX/50 ローカルバス機
INTEL486DX33MHz	INTEL486DX50MHz (冷却ファン付)	INTEL486DX50MHz (冷却ファン付)
256KB (MAX)		256KB (MAX)
4MB (MAX32MB)		8MB (MAX64MB)
170MB		245MB
————		CT−803 × 1
————		DC−660/16MB RAM 付 (LOCALBUS)
AT バス × 7		ローカルバス × 3 · AT バス × 5
200W 50/60Hz 100〜120V·220〜240V		230W 50/60Hz 100〜120V·220〜240V
左ページと同じ		左ページと同じ
ET−4000S−VGA 1280 × 1024(16 色)、1024 × 768(256 色) 800 × 600(256 色)1MB RAM 付		LOCAL BUS用 ET-4000使用 ENHANCED VGA 1280 × 1024(16 色) 1024 × 768(256 色) 800 × 600(65535 色) 640 × 480(65535 色)
MT: ミニタワー (D415 × W200 × H345)		FT: フルタワー (D445 × W195 × H650)
ET−486DX33/1704 **¥349,000**	ET−486DX50/1704 **¥419,000**	**CALL**

●各種ソフトウェア●

ディスプレイディスパッチドライバー
(D·D·D)英語ドライバー日本語化ソフト…特価 ¥ 3,800
DOS/V5.0 …特価
日本語 WINDOWS3.01 …特価
MS-WORKS …特価
MS-WORD …特価 CALL
MS-EXCEL …特価
LOTUS1-2-3/WINDOWS …特価
CARD3+ …特価
同事例集(ビジネス編、スクール編、バラエティ編) …特価
Mike Dytka Football VGA …特価 ¥11,000
Flight Sim4.0(3.5") …特価 ¥16,500
FALCON 3.0(3.5") …特価 ¥14,700
MOON BASE …特価 ¥12,000
Wing Commander with 2missions(CD-ROM) …特価 ¥22,100

●ジョイスティック●

超お買い得スティック ¥3,980 ヨリ各種あります。
Gravis Joy Stick …特価 ¥ 9,900
Thrust master …特価 ¥26,900
Thrust master Joy Stick …特価 ¥24,900
CLUSTER Stick E's (SEGA/ メガドライブ・テラドライブ専用) …特価 ¥ 4,800

●各種アクセサリー●

CONNER HDD(IDE·SCSI Ⅱ)
QUANTUM HDD(IDE)
40MB 〜 360MB …特価 ¥28,000 ヨリ
TEAC FDD
3.5"(1.44MB) …特価 ¥14,900
5.25"(1.2MB) …特価 ¥14,900
DUAL FLOPPY DISK DRIVE
3.5"+5.25" 一体型 …特価 ¥35,700
ASTEC DELTA(電源器)
200W …特価 ¥17,600
300W …特価 ¥CALL
ミニタワーケース
ET−100 ¥16,500
KB−110 ¥18,500
ミドルタワーケース
TB−052 ¥23,500
フルタワーケース
TC−052 ¥29,500

●増設 RAM●

1MB SIMM(70ns)9Bit …特価 ¥ 6,300 ヨリ
4MB SIMM(70ns)9Bit …特価 ¥25,000 ヨリ

●S-VGA カード●

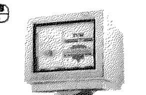
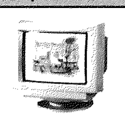
Tseng Lab 純正 ET-4000 Hi-COLOR 65535 色 NEW RAM DAC 付 (RAM1MB) …特価 ¥35,000
ORCHID 社製 S3 チップ使用 Hi-COLOR 32768 色 Fahrenheit 1280 (RAM1MB) …特価 ¥45,800
Diamond 社製 S3 チップ使用 Hi-COLOR 32768 色 STALTH/VRAM(RAM1MB) …特価 ¥46,500
CACHE VGA 1280×1024 256色NCRのアセクラレータ付 DOSもWINDOWS も高速! (RAM2MB) …特価 ¥55,000
OAK オークコンポ 067 1024 × 768 βバージョン日本語ドライバーソフト付(RAM512KB) …特価 ¥25,000
OAK VG-7000(RAM512KB) …特価 ¥12,800

●その他●

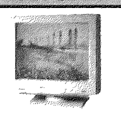
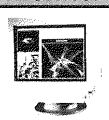
マウス(シリアル) ¥3,980 ヨリ
101 キー英語キーボード ¥6,980
トラックボール・各種ケーブル・モジュラーリンク·OA クリーナー・パソコンデスク・クリーニングディスケット等など便利なアクセサリーいろいろ展示中。

TVM〈COLOR Monitor〉TATUNG

大量仕入 超特価提供!

0.28/15	0.28/14
 フラットスクウェア ブラウン管	
Media Scan 5A + MD-15	ULTRA Super VGA CM-14UAS
0.28m/m·1,024×768·15 インチフラット	0.28m/m·1,024×768·14 インチ
NON-INTERLACED(ノンインターレス)	NON-INTERLACED(ノンインターレス)
65MHz/27〜57KHz/40〜100KHz ノングレア ·Flicker Free	65MHz/31.5/35.5/37.5/48KHz 50〜90Hz/ノングレア/Tinted
メーカー希望小売価格 ¥168,000	メーカー希望小売価格 ¥118,000
¥ 99,000	**¥ 65,000** 45%OFF

TATUNG Color Monitor

0.31/20	0.28/17
	
Multi-Scan CM20MBD	Multi-Scan CM-17MBD28
0.31m/m·1,280 ×1,024·20 インチ	0.28m/m·1,280 × 1,024·17 インチ
NON-INTERLACED(ノンインターレス)	NON-INTERLACED(ノンインターレス)
100MHz/29〜65KHz/50〜90KHz Mac Ⅱにそのまま使用できます.	100MHz/29〜65KHz/50〜90Hz Mac Ⅱにそのまま使用できます!
メーカー希望小売価格 ¥498,000	メーカー希望小売価格 ¥298,000
¥290,000 41%OFF	**¥170,000** 43%OFF

CM−17MBD26 (0.26m/m) もあります。 ¥190,000

GROWING UP マザーボード

BABY-AT マザーボード

新設計EFA社製 34M33/50 マザーボードは CPU を乗せ換えるだけで 386 から 486DX50MHz にグレードアップできます。



CPUなし、386,486共通
AMI BOIS,128KBキャッシュ付
EFA社34M33/50 マザーボード(AT バス × 7)
定価 ¥78,000 → 特価 **¥59,000**
今までの普通の 386DX マザーボートもあります。 超特価放出 ¥29,000

お願い!ご注文の際は必ずお電話にて在庫、価格、納期のご確認をお願いします。
(価格、仕様が変更になる場合がございます。)
当社取扱パソコン、カラーモニターは日本到着後全品連続 48 時間の検査後に出荷しております.

* 全商品消費税、送料別の価格です.
* 銀行振込:協和埼玉銀行 外神田支店 普通 1047406 (株)クラスター

★各種カードご利用できます。 大歓迎!最高60回払!
→ ご来店の方はもちろん、通信販売でもご利用いただけます。 ★企業の方へ…リースも承ります。
(カードの場合は価格が異なりますのでお電話にて、お問い合わせ下さい。)

Dyna Star パソコン日本総発売元 TEKRAM 日本総代理
ソフトウェアー、ハードウェアー開発

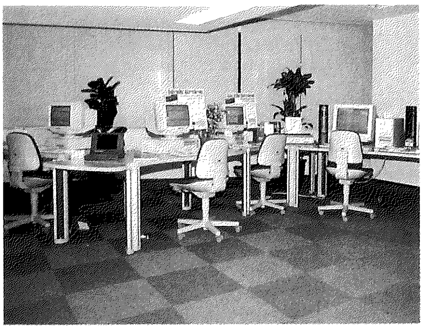
株式会社クラスター

フレンドリーショップ募集中!

〒101 千代田区外神田4-4-3 OGISOビル6F TEL/03(3257)3261 FAX/03(3257)3850

〈資料請求番号007〉

FORUM
千葉電子PBショールーム



豊富な製品ラインアップ

米国パッカード・ベル社の正規代理店である千葉電子は，286から486までの幅広い製品ラインアップと，ショールームにおけるソフトウェア動作確認などによって，ユーザーから信頼されるビジネスを築こうとしている．ショールームには，パッカード・ベルのマシンがずらりと並ぶ，幅広い品揃えというだけあって，CPUの種類だけでなくノート型からミニタワーまでさまざまなスタイルのパソコンが展示されている．Windowsを使おうというユーザーには，格安な17インチモニターが魅力だろう．

「パソコンをどう使うかはお客さん次第ですから，お客さんの使いたいソフトが動かなければ意味がないわけです．ショールームでは，実際にソフトが動くかどうかを確認していただけます．また，目的に合わせて適切なマシンを選んでいただくこともできます」と，話すのは千葉電子の鳥飼主任．人気のある486マシンばかりを買えばいいというものではないというわけだ．米国製マシンとユーザー本意の姿勢でDOS/V勢の中でも独自の立場を貫いている．

DOS/Vという製品

最初に登場したDOS/Vパッケージは，DOS/Vを提唱した日本IBMが出荷したIBM DOS J4.0/V．その後，バージョンアップされ，IBM DOS J5.0/Vになっている．現状では，互換機を購入したユーザーも，日本IBMのDOS/Vパッケージを買って利用しているのが一般的だ．

現在，出荷あるいは発表されているDOS/VのOS製品は次の通り．

IBM DOS J5.0/V
(日本IBM)
COMPAQ MS-DOS 5.0J/V
(コンパック)
DR DOS 6.0/V
(デジタル・リサーチ・ジャパン)

コンパックのDOSは，マイクロソフトからOEM供給を受けた初めてのDOS/Vになる．キャラクタ系デバイスドライバを組み込み・切り放すADDDRV，DELDRVコマンドが付属していること，日本語環境から英語環境のソフトを起動させたときに表示できないという問題を回避するVE386.EXEを用意していることなど，IBMのDOS/Vとは異なる点も多い．

また，DR DOS 6.0/Vは，MS-DOS互換OSの日本語版とあって，ハードディスクの容量とパフォーマンスを向上させるDiskMAXなど，さまざまな機能が付属している．この製品は，DOS/Vマシンメーカーに OEM供給される．

VGA

IBM PS/2に搭載されているビデオシステム．サードパーティーからPC/AT用にVGA互換ボードが発表されてから急速に普及し，米国などでは標準的な位置を占めている．DOS/Vは，VGAを搭載しているPCまたはPS/2においてソフト的に日本語環境を実現する．つまり，DOS/Vの基本はVGAにあるというわけ．

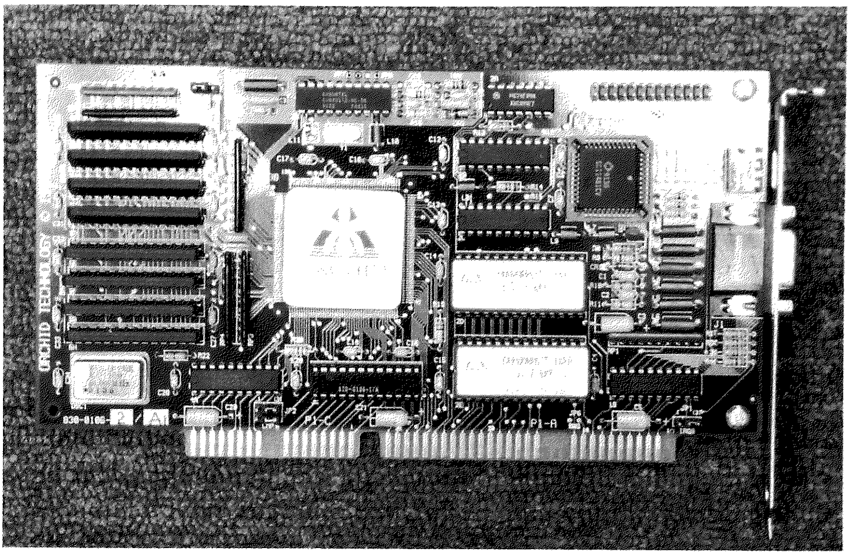
高解像度Windows

なぜ，DOS/Vが注目されているかといったら，Windowsにおいて高解像度表示を手軽に実現できるからだと言っていい．海外で販売されているビデオカードを装着すれば，あとはソフトウェアの設定を変更するだけで，日本語Windowsが高解像度で利用できるようになる．従来は，ハードウェアがあっても日本語Windows用のデバイスドライバがなければ日本語で使えなかったのだが，DDD(Display Dispatch Driver)によってビデオカードとそのWindows用デバイスドライバ(英語版)さえあれば日本語Windowsで使えるようになった．800×600ドット，1024×768ドットのSVGAなどを実現できる．それ以上の，1280×960や2048×2048という高解像度も可能になった．表示色も16，256，3万2000，1600万色までハードウェア次第で可能だ．

ビデオカード

画面表示用の拡張ボード．IBM PC系のパソコンでは，ビデオシステムは可変になっているため，非常に多くのビデオカードが流通している．現在使われているのは，VGAに互換をとりながらVGA以上の解像度や色数を実

Fahrenheit1280ボード



現しているSVGAと呼ばれるものが主流だ．いま，注目されているのはWindows用のアクセラレーターを搭載しているビデオカード．Windowsの描画ルーチンをハードウェア化しているので，Windowsの実行が10倍も速くなるといわれる．「S3のビデオカード」などと呼ばれるが，ビデオカードの呼び方は混乱を招いている．ビデオカードの機能は搭載するチップセットによって決まる．そして，同じチップセットを搭載したカードが幾つものメーカーから発売される．「S3のビデオカード」と呼ばれているものは，S3というメーカーの出している86C911というチップを搭載したビデオカードを意味している．86C911を搭載したカードは幾つも出ており，有名なものとして，Orchidというメーカーの Fahrenheit1280がある．

FORUM T-ZONE



PS PLAZAとIBMゾーンの２フロアを持つT-ZONEは，DOS/V関連のハードウェア，ソフトウェアとも充実した品揃えを誇っている．ハードウェアは，PS/55noteをはじめとする純正DOS/Vマシンと互換機を揃え，VGAアダプタカードや大型のディスプレイなど，DOS/Vデパートといってもいい多彩さだ．また，サウンドボードとCD-ROMドライブ，WindowsMMEなどのCD-ROMソフトウェアをセットにしたマルチメディアアップグレードキットを，ファーストロットでいち早く国内に紹介したのもT-ZONEだ．DOS/Vの世界に入ったユーザーの「あれもやりたい，これもつけたい」という気持ちに応えてくれる．
ソフトウェアもゲームから開発ツールにいたるまで品揃えが豊富．常時3000本以上もの製品を揃えている．米国で発売されたソフトの入荷が早い店として，メーカーの技術者などの来店も多いという．

ADO TERADRIVE ゲーム作家新人賞コンテスト

TERADRIVEを使ったゲームコンテスト どしどし作ってご応募下さい。
★受付はT・ZONE各店にて

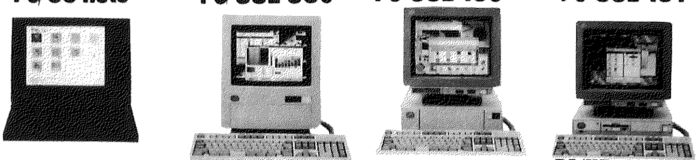
主催：㈱亜土電子工業
期間：平成4年5月1日～平成4年7月31日迄
発表：平成4年8月31日

《賞品》

1位 最優秀賞 アメリカ西海岸ツアー旅行（副賞50万円）
2位 優秀賞 ハワイ旅行 （副賞30万円）
3位 佳作 香港旅行 （副賞20万円）
4位 特別賞 TERADRIVE

IBM NEWマシンぞくぞく大量入荷!!

PS/55 note　PS/55Z 30U　PS 55Z 10S　PS 55Z 10T



大容量80MBハードディスク内蔵（S18 SV8 / S28 SA8）
パワフルなDOS/Vノート
80386SX（16MHz）2MB FDD内蔵

XGA採用パーソナルディスクトップ
80386SX（20MHz）2MB or 6MB
パソコンらしくないパソコン？
オシャレ

386SX搭載の
スタンダードPC
DOSV、VGA、ATバス採用
80386SX（20MHz）2MB or 6MB

PS/55Z
最強パフォーマンスPC
XGA高解像度・大容量
80386（20MHz）2MB or 6MB

5530U用SCSI HD増設キット

SCSI II/A + ケーブルキット + CONNER SCSIドライブ

120MB、19mS	超特価	¥178,000
210MB、12mS	超特価	¥248,000
360MB、12mS	超特価	¥328,000
540MB、12mS	超特価	¥410,000
510MB、6mS	超特価	¥578,000

5523SJ4 限定 超特価¥168,000

増設用RAM

5530U	2MB	¥21,000
5530U	4MB	¥40,000
5530U	8MB	¥80,000
5510S	2MB	¥25,000

NEW 新型高性能32ビットマイクロ・プロセッサー386SLC搭載

180%のスピード（386SX/20対比）

5530UAI（386SLC/20MHz、FDモデル） ¥540,000
5530UA4（386SLC/20MHz、40MBHDモデル） ¥620,000
5530UAW（386SLC/20MHz、80MBHD WINDOWSモデル） ¥798,000

180%のスピード（386SX/16対比）

8551-SZ8（386SLC/16MHz、OS/2、80MBHD、RAM12MB、マウス） ¥664,000
8551-SV8（386SLC/16MHz、DOS/V、80MBHD、4MB） ¥514,000

5530U（386SX/20）T・ZONEモデル/スペシャルプライス

メモリー＼HDD	120MB	210MB	360MB	540MB
2MB	495,000	530,000	598,000	638,000
4MB	505,000	540,000	608,000	648,000
6MB	515,000	550,000	618,000	658,000
8MB	525,000	560,000	628,000	668,000
10MB	535,000	570,000	638,000	678,000
12MB	545,000	580,000	648,000	688,000
14MB	555,000	590,000	658,000	698,000

※上記セットをお買い上げのお客様だけのスペシャルオファー
日本語Windows 3.0とADO BLACK MOUSE 特別価格¥20,000
日本語Windows 3.0とIBM純正 MOUSE 特別価格¥25,000
※HARD DISK はCONNER のSCSI用HARD DISK.(1年保証)。
MEMORY は5年保証です。
SCSI BOARD はIBM のSCSI II(32ビット)です。
(32ビットBUSMA USTER 512K CACHE)

WINDOWS

S3より速い

◎最高速グラフィックスボード
限定 ATI GRAPHICS ULTRA（1MB） 超特価¥89,800
ATI GRAPHICS VANTAGE（1MB） 超特価¥59,800
◎D.D.Dディスプレーディスパッチドライバー 特価¥3,800
英語ドライバーで1024×768の高速日本語WINDOWSが使えます。
◎TR-OIJシリアルトラックボール 特価¥5,800
◎ADO BLACK MOUSE ¥6,000
◎NANAO T660i 20"トリニトロン1280×1024 特価¥498,000
◎TATUNG 17" 0.28mm 1280×1024 特価¥180,000
◎TATUNG 17" 0.26mm 1280×1024 特価¥198,000

AST RESEARCH JAPAN K.K.

ASTから DOS/Vマシン
80486/33MHz
FDDモデル ~~¥308,000~~
40MBモデル ~~¥355,000~~



● 高速32ビットCPUを採用。メインメモリ4MBを標準搭載。
● Weitek 4167数値演算プロセッサの搭載が可能。
● IBM PC ATスロットを5基装備。

PCvision

4/33 モデル3V ¥361,000
モデル83V ¥432,000
モデル83B ¥610,000

386SX/20
モデル3V ¥308,000
モデル43V ¥355,000
モデル43SP ¥469,000

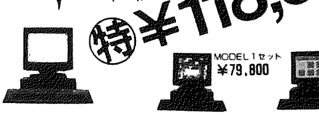
Premium Exec 386SX/25
モデル63V/4 ¥414,000
モデル83V/4 ¥457,000
モデル123V/4 ¥528,000

Premium Exec 386SX/25C
モデル63V/4 ¥614,000
モデル83V/4 ¥657,000
モデル123V/4 ¥729,000

TERADRIVE

専用カラーディスプレイが（標準価格79,800円）いっしょについてこの値段

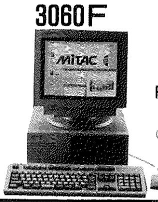
超特価 MODEL 3セット 特¥118,000



MODEL 1セット ¥79,800
MODEL 2セット ¥89,800

MiTAC 3060F

入門に最適！
WINDOWS最適！
S3標準搭載！



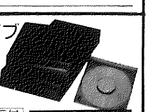
◎60MBモデル RAM 4MB、3.5"＆5" FD、S3＆Hi-Color+SVGAモニタ 超特価¥298,000
◎120MBモデル RAM 6MB、3.5"＆5" FD、S3＆Hi-Color+SVGAモニタ 超特価¥348,000

CREATIVELABS SOUND BLASTER

マルチメディア対応のSOUND BLASTERは必需品です！ 特価提供中

これ1つでマルチメディアが楽しめます。 内蔵CD-ROM SOUND BLASTER PRO MIDI KIT マルチメディアイクステンション他ソフト付 UPGRADE KIT ~~¥164,800~~	ステレオ22FM音源 CD-ROMインターフェイス他 SOUND BLASTER互換 Multimedia Sound Standard PRO ~~¥57,800~~
11FM音源 MIDIジョイスティックパワーアンプマイク入力 各種ソフト付 V2.0 ~~¥34,800~~	PS/55等のマイクロチャンネル版です。 MCA ~~¥64,800~~

マルチメディア対応高速CD-ROMドライブ



● 標準SCSI（最大7台まで同時接続可能）
● 高速アクセス（340ms）、64KBキャッシュ内蔵
● CD-ROM XA対応
● 市販オーディオディスク演奏可能

	ドライブのみ	AT I/F付	PS/2 I/F付
内蔵モデル	¥125,000	¥145,000	¥158,000
外付モデル	¥135,000	¥155,000	¥168,000

特価提供中

the Manhole （日本語版）

それはマンホールからはじまった……
ゲームを超えたファンタジーアドベンチャー発売中！
DOS/V対応
IBM PS/55
テラドライブ対応
¥9,800



DOS/V版 パソコン電子手帳くん ¥14,800

新登場

シャープの電子手帳とDOS/Vマシンの間でデータのやりとりができるようになりました。
対応している電子手帳
PA6000、6500、7000、7500、8500、8600シリーズ。

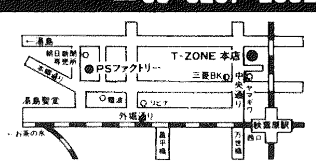
T・ZONE 新宿店 ☎03-3342-5533

1F APPLE＆NEC 03-3342-1901
2F IBM PS/FACTORY 03-3342-5533
FAX 03-3342-1903



〒160 東京都新宿区西新宿1-12-18

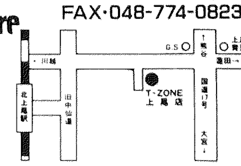
T・ZONE 本店 ☎03-3257-2652



〒101 千代田区外神田4-4-1

T・ZONE 上尾店 ☎048-774-0511

Super Store AGEO オープン！
FAX・048-774-0823
営業時間 AM10:30～PM7:00



〒362 埼玉県上尾市緑丘3-5-28

☆T・ZONEの7Fは専門書籍（国内・外）・データーブックの専門フロアです。大量在庫中！ ほしい本は何でも揃います。

〈資料請求番号008〉

PC-9801・DOS/V エスケープシーケンスキャラクター一覧

ESC は1BHを表します。
イタリック体の*pc*，*pl*，*pn*，*ps*はパラメータでそれぞれの値が入ります。

■PC-9801シリーズ

ESC [*pl*;*pc*H　Direct cursor addressing
カーソルを*pl*行目の，*pc*カラム目に移動する．*pl*が最大行より大きい値が指定された場合は*pl*に最大行が，*pl*が0または省略された場合は先頭行が指定されたとして処理される．同様に*pc*が最終カラムより大きい値が指定された場合は最終カラムが，*pl*が0または省略された場合は先頭カラムが指定されたとして処理される．

ESC [*pl*;*pc*f　ESC [*pl*;*pc*Hと同様の処理を行う

ESC =*lc*　ESC [*pl*;*pc*Hと同様の処理を行う
ただしパラメータの与え方は2進数で，20Hのオフセットが加えられた値となる．たとえば*l*は行位置であり1行目を指定するには20Hとなり，また*c*はカラム位置であり，1カラム目を指定するには20Hとなる．*l*と*c*のパラメータは省略できない．

ESC [*pn*A　Cursor up
カーソルを同じカラム位置のまま，行を*pn*だけ減じた値とする．カーソルが先頭行にある場合，減じた値が1より小さい場合はカーソル位置を先頭行とし，またパラメータを省略した場合，*pn*の値が0の場合は行の値を1だけ減じたとして処理する．

ESC [*pn*B　Cursor down
カーソルを同じカラム位置のまま，行を*pn*だけ加えた値とする．カーソルが最大行にある場合，加えた値が最大行より大きい場合はカーソル位置を最大行とし，またパラメータを省略した場合，*pn*の値が0の場合は行の値を1だけ加えたとして処理する．

ESC [*pn*C　Cursor forward
カーソルを同じ行のまま，カラム位置を*pn*だけ加える．カーソルが最終カラムの場合，加えた値が最終カラムより大きい場合は，カーソルを最終カラムとし，またパラメータを省略した場合，*pn*の値が0の場合はカラム位置の値を1だけ加えたとして処理する．

ESC [*pn*D　Cursor backward
カーソルを同じ行のまま，カラム位置を*pn*だけ減じた値とする．カーソルが先頭カラムにある場合，減じた値が1より小さい場合はカーソルを先頭カラムとし，またパラメータを省略した場合，*pn*の値が0の場合はカラム位置の値を1だけ減じたとして処理する．

ESC [0J　Clear from cursor to end of screen
カーソル位置から，最大行の最終カラムまでをクリアする．カーソル位置は変わらず，パラメータ0は省略可能．

ESC [1J　Clear from beginning of screen to cursor
ホームポジション（先頭行の先頭カラム）から，カーソル位置までをクリアする．カーソル位置は変わらない．

ESC [2J　Clear screen
スクリーンをクリアする．カーソルはホームポジションになる．

ESC ＊　ESC [2Jと同様の処理を行う

ESC [0K　Clear from cursor to end of line
カーソル位置から，行の右端までをクリアする．カーソル位置は変わらず，パラメータ0は省略可能．

ESC [1K　Clear from beginning of line to cursor
行の先頭からカーソル位置までをクリアする．カーソル位置は変わらない．

ESC [2K　Clear entire line containing cursor
カーソルの存在する行をすべてクリアする．カーソル位置は変わらない．

ESC [*pn*M　Delete line
カーソル位置から下に*pn*行削除し，以降の行をスクロールアップする．カーソル位置は変わらない．パラメータを省略するか0が指定された場合は*pn*は1が指定されたとして処理する．

ESC [*pn*L　Insert line
カーソル位置の行を*pn*行だけ下に移動する．移動する行が最大行を越えた場合，その行は失われる．カーソル位置は変わらない．パラメータを省略するか0が指定された場合は*pn*は1が指定されたとして処理する．

ESC D　Index
カーソルと同じカラム位置で1行下に移動する．カーソルが最終行の場合は1行スクロールアップする．

ESC E　Next line
カーソルを1行下の行の左端に移動する．カーソルが最終行にある場合は，1行スクロールアップする．

ESC M　Reverse index
カーソルを同じカラム位置で1行上に移動する．カーソルが先頭行にある場合は1行スクロールダウンする．

ESC [s　Save cursor position
カーソル位置（行，カラム）とその表示文字の属性を退避する．

ESC [u　Set cursor position
ESC [sで退避した内容を戻す．以前にESC [sが実行されていない場合は，カーソルはホームポジションに移動し，属性は既定値となる．

ESC [6n　Cursor position report
カーソル位置を直後のコンソール入力呼び出しにて知らせる．その形式はESC [*pl*;*pc*Rとなる．

ESC) 0　Select kanji mode
漢字を扱うモードを指定する．このモードになっていないと，漢字を入力しても漢字の表示は行われない．システムの既定値はこのモードとなっている．

ESC) 3　Select graph mode
グラフィック文字を扱うモードを指定する．このモードによってグラフィック文字の入力および出力が可能となる．

ESC [>5l　Enable cursor display
カーソルを画面上に表示するモードを指定する．システムの既定値はこのモードとなっている．

ESC [>5h　Disable cursor display
カーソルを画面上に表示しないモードを指定する．このモードになるとそれ以降ESC [>5lが指定されるまで画面上にカーソルは表示されない．

ESC [>1h　Enable bottom line
スクリーンの最下行をプログラムで使用可能にするモードを指定する．これによりスクリーンのすべてがプログラムで使用できる．

ESC [>1l　Disable bottom line
スクリーンの最下行をシステムで使用するモードを指定する．システムの既定値はこのモードとなっている．

ESC [>3h　Select 20 line mode
スクリーンの表示行数を20行に指定する．（ノーマルモードのみ）

ESC [>3n　Select 31 line mode
スクリーンの表示行数を31行に指定する．（ハイレゾモードのみ）

ESC [>3l　Select 25 line mode
スクリーンの表示行数を25行に指定する．システムの既定値はこのモードとなっている．

ESC [*ps*;…;*ps*m　Character attribute
表示文字の属性を指定する．属性は一度指定すると以降に続く表示文字に適用され，次の属性の指定まで有効となる．パラメータの指定は一度に複数指定できる．パラメータを省略した場合は*ps*に0が指定されたとして処理する．

0		規定の属性	
1		ハイライト（モノクロのみ）	
2		バーチカルライン	
4		アンダーライン	
5		ブリンク	
7		リバース	
16	8	シークレット	
	30	黒（カラーモード時）	淡（暗）（モノクロモード時）
18	34	青	↑
17	31	赤	
19	35	紫	
20	32	緑	
22	36	水色	
21	33	黄色	↓
23	37	白	濃（明）
	40	黒	
	41	青	
	42	赤	
	43	紫	
	44	緑	
	45	水色	
	46	黄色	
	47	白	

ESC [*pn*;…;*pn*p　Keyboard reassignment
キーの割当を変更する．ESC [に続く最初の1文字に対応するキーに，2番目以降の文字または文字列を割り当てる．

ESC ["string";p　ESC [*pn*;…;*pn*pと同様の処理を行う

ESC [*pn*;"string";*pn*p　ESC [*pn*;…;*pn*pと同様の処理を行う

■DOS/V

ESC [*pl*;*pc*H　カーソル位置（CUP）
カーソルを*pl*行目の，*pc*カラム目に移動する．*pl*に最大行より大きい値が指定された場合は*pl*に最大行が，*pl*が0または省略された場合は先頭行が指定されたとして処理される．同様に*pc*に最終カラムより大きい値が指定された場合は最終カラムが，*pl*が0または省略された場合は先頭カラムが指定されたとして処理される．

ESC [*pn*A　カーソル上向き移動（CUU）
カーソルを同じカラム位置のまま，行を*pn*だけ減じた値とする．カーソルが先頭行にある場合，減じた値が1より小さい場合はカーソル位置を先頭行とし，またパラメータを省略した場合，*pn*の値が0の場合は行の値を1だけ減じたとして処理する．

ESC [*pn*B　カーソル下向き移動（CUD）
カーソルを同じカラム位置のまま，行を*pn*だけ加えた値とする．カーソルが最大行にある場合，加えた値が最大行より大きい場合はカーソル位置を最大行とし，またパラメータを省略した場合，*pn*の値が0の場合は行の値を1だけ加えたとして処理する．

ESC [*pn*C　カーソル前方移動（CUF）
カーソルを同じ行のまま，カラム位置を*pn*だけ加える．カーソルが最終カラムの場合，加えた値が最終カラムより大きい場合は，カーソルを最終カラムとし，またパラメータを省略した場合，*pn*の値が0の場合はカラム位置の値を1だけ加えたとして処理する．

ESC [*pn*D　カーソル後方移動（CUB）
カーソルを同じ行のまま，カラム位置を*pn*だけ減じた値とする．カーソルが先頭カラムにある場合，減じた値が1より小さい場合はカーソルを先頭カラムとし，またパラメータを省略した場合，*pn*の値が0の場合はカラム位置の値を1だけ減じたとして処理する．

ESC [*pl*;*pc*f　水平および垂直位置（HVP）

ESC [*pl*;*pc*R　カーソル位置報告（CPR）

ESC [6n　装置状態報告（DSR）
カーソル位置を直後のコンソール入力呼び出しにて知らせる．その形式はESC [*pl*;*pc*Rとなる．

ESC [s　カーソル位置保管（SCP）
カーソル位置（行，カラム）とその表示文字の属性を退避する．

ESC [u　カーソル位置復元（RCP）
ESC [sで退避した内容を戻す．以前にESC [sが実行されていない場合は，カーソルはホームポジションに移動し，属性は既定値となる．

ESC [2J　画面消去（ED）
スクリーンをクリアする．カーソルはホームポジションにとなる．

ESC [K　ライン消去（EL）
カーソル位置から，行の右端までをクリアする．カーソル位置は変わらない．

ESC [*ps*;…;*ps*m　グラフィック属性の設定（SGR）
表示文字の属性を指定する．属性は一度指定すると以降に続く表示文字に適用され，次の属性の指定まで有効となる．パラメータの指定は一度に複数指定できる．パラメータを省略した場合は*ps*に0が指定されたとして処理する．

0	規定の属性
1	ハイライト（単色グラフィックモードはサポートしない）
4	アンダーライン（単色文字モードのみ）
5	ブリンク（英語モードの文字モードのみ）
7	リバース（文字モードのみ）
8	シークレット（単色グラフィックモードはサポートしない）
30	黒の文字（カラーモードのみ）
31	赤の文字（カラーモードのみ）
32	緑の文字（カラーモードのみ）
33	黄色の文字（カラーモードのみ）
34	青の文字（カラーモードのみ）
35	紫の文字（カラーモードのみ）
36	水色の文字（カラーモードのみ）
37	白の文字（カラーモードのみ）
40	黒の背景（カラー文字モードのみ）
41	青の背景（カラー文字モードのみ）
42	赤の背景（カラー文字モードのみ）
43	紫の背景（カラー文字モードのみ）
44	緑の背景（カラー文字モードのみ）
45	水色の背景（カラー文字モードのみ）
46	黄色の背景（カラー文字モードのみ）
47	白の背景（カラー文字モードのみ）

ESC [＝*pn*h　モードの設定（SM）
*pn*で指定された画面モードに画面を設定する

0～2	予約済み
3	80×25カラー文字モード（CGA属性）
4～6	予約済み
7	行末で折り返し（行末を越えて入力すると新しい行がはじまる）
13～16	予約済み
17	640×480（80×30）単色グラフィックモード
18	640×480（80×30）カラーグラフィックモード
19	予約済み
114	640×480（80×30）カラーグラフィックモード
115	80×25カラー文字モード（拡張CGA属性）

ESC [＝h　ESC [＝*pn*hと同様の処理を行う

ESC [＝0h　ESC [＝*pn*hと同様の処理を行う

ESC [?7h　ESC [＝*pn*hと同様の処理を行う

ESC [＝*pn*l　モードのリセット（RM）
パラメータ7が行末の折り返しをリセットするのを除いてSMと同じ．行末を越えた文字は切捨てられる．

ESC [＝l　ESC [＝*pn*lと同様の処理を行う

ESC [＝0l　ESC [＝*pn*lと同様の処理を行う

ESC [?7l　ESC [＝*pn*lと同様の処理を行う

ESC [*pn*;…;*pn*p　キー再割り当て

ESC [0q　拡張機能キーボード上での拡張キーの再割り当て禁止
拡張キーボード上での拡張キーの再割り当てを不可能にする．

ESC [1q　拡張機能キーボード上での拡張キーの再割り当て許可
拡張キーボード上での拡張キーの再割り当てを可能にする．このシーケンスは/Xパラメータを指定するのと同様な効果を持つ．

MiTAC



高解像度ビデオアダプタ標準装備。最高1024×768×256色対応。
ビデオメモリ1MB標準装備。
高解像度 Windows 3.0ドライバ標準装備

フルサイズPC/AT拡張スロット4スロット装備

Weitek演算プロセッサソケット標準装備

高性能486DX-33MHz CPU搭載。
80387数値演算コープロセッサ内蔵。

標準4MBシステムメモリ、最大32MBまで拡張可能。

日本語106キーJISキーボード標準装備

ノンインターレーススーパーVGA、XGA対応高解像度モニタM1458（オプション）

5.25インチドライブベイに各種オプション搭載可能。5.25インチFDDドライブオプション（2ドライブ装備時には任意のフロッピードライブからの起動設定をソフトウェアでセット可能）

intel init

ハードディスクのオプションとして最高500MBHDD内蔵可能。

64KB外部キャッシュメモリ搭載。

バイリンガルOS DOS J5.0/V対応。

マイタックMM-PSマウス標準装備

4060G ベースモデル 4060G-003-4(JP) ￥498,000

OADG
PC Open Architecture Developers' Group

マイタック4060シリーズ新登場

これからのベーシック 486。

いま大きく変化しているパーソナルコンピュータ環境。キャラクターベースからビジュアルなグラフィックスへ。制約された基本メモリ領域からより広大な拡張メモリエリアへ。より高速、高性能なCPUへ。性能、価格すべてが快適な環境を求めています。大きく変動しているパーソナルコンピュータ環境を常に快適に利用できるシステムをマイタックは提供します。
全モデルの基本機器構成を充実しました。

- OADG仕様準拠
- 高性能32ビットCPU装備
- 高容量システムメモリ装備
- 高速キャッシュメモリ装備
- 高解像度ディスプレイアダプタ装備
- 日本語106キーJISキーボード装備
- 高品位マウス装備

仕様 ＼ モデル	New 4060Gシリーズ	New 4060Fシリーズ	Price OFF 3060Gシリーズ	New 3060Fシリーズ
				
プロセッサ	486DX-33	486SX-25	386DX-33	386SX-25
システムメモリ	システムメモリ4MB(32MBまで拡張可能)			システムメモリ4MB (16MBまで拡張可能)
マウス	マイタックMM-PSマウス標準装備			
ビデオアダプタ	1,024×768×256色			1,024×768×16色
60MBハードディスクモデル	4060G-063-4(JP) ¥568,000	4060F-063-4(JP) ¥498,000	3060G-063-4(JP) ¥428,000	3060F-063-4(JP) ¥358,000
120MBハードディスクモデル	4060G-123-4(JP) ¥618,000	4060F-123-4(JP) ¥548,000	3060G-123-4(JP) ¥478,000	3060F-123-4(JP) ¥408,000

※intel in it は米国インテル社の商標です。その他の会社名、製品名は各社の商標または登録商標です。

株式会社 マイタック ジャパン
〒141 東京都品川区北品川5−7−14
TEL. 03(5420)2822 FAX. 03(5420)4586

〈資料請求番号 P03〉